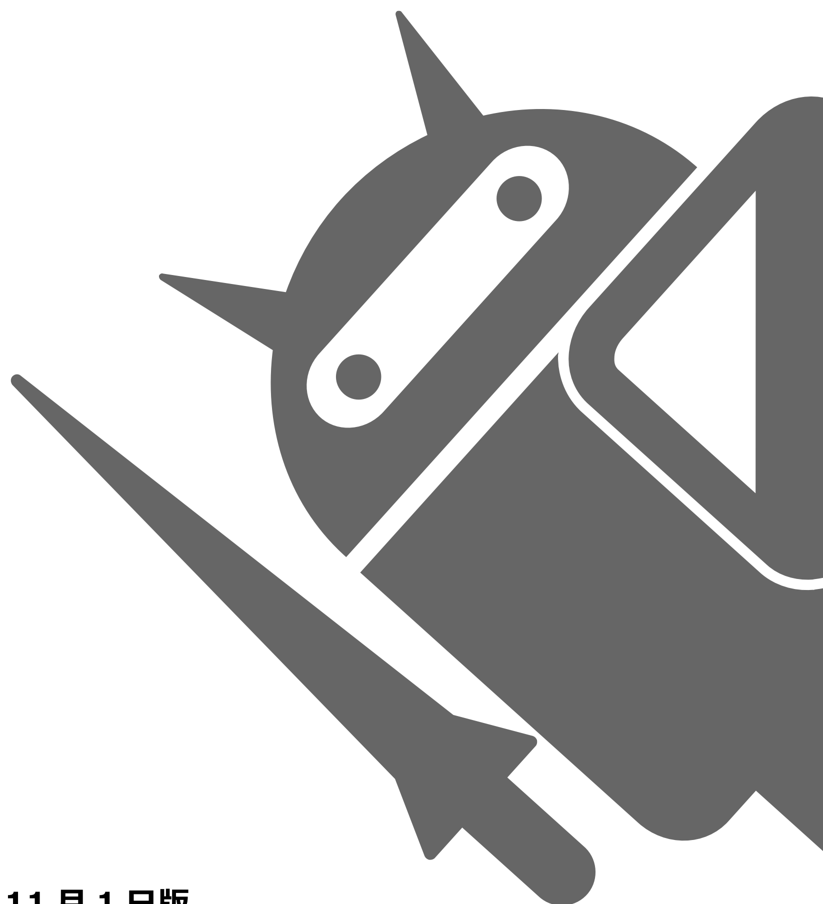
JAPAN
SMARTPHONE
SECURITY
ASSOCIATION

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 11 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

文書管理番号：JSSEC-TECA-SC-GD20121101β

※　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™は、Google, Inc.の商標または登録商標です。また、本文書に登場する会社名、製品名、サービス名は、一般に各社の登録商標または商標です。本文中では®、TM、© マークは明記していません。

※　この文書の内容の一部は、Google, Inc.が作成、提供しているコンテンツをベースに複製したもので、クリエイティブ・コモンズの表示 3.0 ライセンスに記載の条件にしたがって使用しています。

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

【β版】

2012 年 11 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

Android アプリのセキュア設計・セキュアコーディングガイド 1

1. はじめに 6

1.1. スマートフォンを安心して利用出来る社会へ 6

1.2. 常にβ版でタイムリーなフィードバックを 7

1.3. 本文書の利用許諾 8

1.4. 2012 年 6 月 1 日版からの訂正記事について 9

2. ガイド文書の構成 11

2.1. 開発者コンテキスト 11

2.2. サンプルコード、ルールブック、アドバンスト 12

2.3. ガイド文書のスコープ 14

2.4. Android セキュアコーディング関連書籍の紹介 15

2.5. サンプルコードの Eclipse への取り込み手順 16

3. セキュア設計・セキュアコーディングの基礎知識 26

3.1. Android アプリのセキュリティ 26

3.2. 入力データの安全性を確認する 38

4. 安全にテクノロジーを活用する 40

4.1. Activity を作る・利用する 40

4.2. Broadcast を受信する・送信する 82

4.3. Content Provider を作る・利用する 110

4.4. Service を作る・利用する 157

4.5. SQLite を使う 197

4.6. ファイルを扱う 214

4.7. Browsable Intent を利用する 240

5. セキュリティ機能の使い方 243

5.1. パスワード入力画面を作る 243

5.2. Permission と Protection Level 256

5.3. Account Manager で独自アカウントを追加する 282

6. 難しい問題 298

6.1. Clipboard から情報漏洩する危険性 298

All rights reserved © Japan Smartphone Security Association.

更新履歴

日付	改訂内容
2012-06-01	• 初版
2012-11-01	• 下記の構成・内容を見直し拡充致しました 　• 4.1 Activity を作る・利用する 　• 4.2 Broadcast を受信する・送信する 　• 4.3 Content Provider を作る・利用する 　• 4.4 Service を作る・利用する 　• 5.2 Permission と Protection Level
	• 下記の新しい記事を追加致しました 　• 2.5 サンプルコードの Eclipse への取り込み手順 　• 3.1 Android アプリのセキュリティ 　• 4.7 Browsable Intent を利用する 　• 5.3 Account Manager で独自アカウントを追加する 　• 6.1 Clipboard から情報漏洩する危険性
• 新版の公開にあたり、皆様から頂いたご意見・コメントを元に本ガイドの内容を更新しました	

All rights reserved © Japan Smartphone Security Association.

■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
	中口　明彦	Android セキュリティ部
	大内　智美	株式会社 SRA
	大平　直之	株式会社 SRA
	熊澤　努	株式会社 SRA
	関川　未来	株式会社 SRA
	中野　正剛	株式会社 SRA
	比嘉　陽一	株式会社 SRA
	福本　郁哉	株式会社 SRA
	星本　英史	株式会社 SRA
	安田　章一	株式会社 SRA
	八尋　唯行	株式会社 SRA
	吉澤　孝和	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
	竹森　敬祐	KDDI 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
	浅野　徹	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
	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
	池邉　亮志	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
	大谷　三岳	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
	小木曽　純	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
	片岡　良典	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
	吉川　岳流	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
	北村　久雄	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
	谷口　岳	タオソフトウェア株式会社

All rights reserved © Japan Smartphone Security Association.

山川　隆郎　　一般社団法人日本オンラインゲーム協会
石原　正樹　　日本システム開発株式会社
森　靖晃　　日本システム開発株式会社
八津川　直伸　　日本ユニシス株式会社
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）

All rights reserved © Japan Smartphone Security Association.

■2012 年 6 月 1 日版制作者■

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、比嘉 陽一、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、熊谷 裕志、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、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、市川 茂、奥山 謙、佐藤 郁恵、西村 宗晃、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、島野 英司、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)

All rights reserved © Japan Smartphone Security Association.

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開放され、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開放されているため、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洩したり、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。

 All rights reserved © Japan Smartphone Security Association.

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

All rights reserved © Japan Smartphone Security Association.

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ます。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
件名:【コメント応募】Android アプリのセキュア設計・セキュアコーディングガイド 2012 年○月○日版
内容:氏名(任意)/所属(任意)/連絡先 E-mail(任意)/ご意見(必須)/その他ご希望(任意)

 All rights reserved © Japan Smartphone Security Association.

1.4. 2012 年 6 月 1 日版からの訂正記事について

本節では、前版の記事について事実関係と照らし合わせることで判明した訂正事項を一覧にして掲載しています。各訂正記事は、執筆者による継続的な調査結果だけでなく読者の方々の貴重なご指摘を広く取り入れたものです。特に、いただいたご指摘は、本改訂版をより実践に即したガイドとして高い完成度を得るための最も重要な糧となっています。

前版を元にアプリケーション開発を進めていた読者は、以下の訂正記事一覧に特に目を通していただきますようお願いいたします。なお、ここで掲げる項目には、誤植の修正、記事の追加、構成の変更、単なる表現上の改善は含みません。

本ガイドに対するコメントは、今後もお気軽にお寄せくださいますようよろしくお願いいたします。

訂正記事一覧

2012 年 6 月 1 日版の修正個所	本改訂版の訂正記事	訂正の要旨
サンプルコード	サンプルコード	サンプルコードのリソース解放処理において、例外を正しく扱うように修正いたしました。
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）	4.5.2.1 DB ファイルの配置場所、アクセス権を正しく設定する （必須）	SQLiteDatabase#openOrCreateDatabase を使用した場合、Android スマートフォンの機種によっては、他のアプリから読み取り可能な DB ファイルが作成されることが判明しましたので、このメソッドを使用しないよう記事を修正いたしました。
5.2.2.1. 公開 Component は Normal Permission で保護する（推奨）	5.2.2.5 独自定義の Normal Permission は利用してはならない （推奨）	追加調査により、自社アプリ間連携において独自定義 Normal Permission を安全に使用する方法が存在しないことが判明しました。独自定義 Normal Permission の使用を禁止するようにルールを修正いたしました。
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）	5.2.2.3 独自定義 Signature Permission は Component の提供側アプリでのみ定義する（必須）	追加調査により、自社アプリ間連携において独自定義 Signature Permission を利用側アプリで定義した場合、安全性に問題があることが判明しました。独自定義 Signature Permission の使用に関するルールを修正いたしました。
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）	5.2.2.3 独自定義 Signature Permission は Component の提供側アプリでのみ定義する（必須））	未登録の独自定義 Permission は Protection Level が normal として扱われるとしていました。しかし、実際には、未登録の Permission を使用するアプリのインストール時には権限の付与が行われませんので、その旨訂正いたしました。

All rights reserved © Japan Smartphone Security Association.

 All rights reserved © Japan Smartphone Security Association.

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。

All rights reserved © Japan Smartphone Security Association.

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。

なお、サンプルコードおよび記事の内容は特別な記述がない限り Android 2.2(API Level 8)以降を対象にしています。Android 2.1(API Level 7)以前のバージョンにおいては動作確認をしておらず、対策として効果がない場合もありますのでご注意ください。また、対象範囲内のバージョンであっても、組み込んだ端末で動作をご確認の上、ご自身の責任のもとでご利用ください。

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。解説においては簡潔さを心がけており、セキュリティ上考慮すべきポイントを本文中で「**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場合があ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な量ですが、それら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すことができるように、サンプルコードはクラス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。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。

デバッグ用にキーストアファイル debug.keystore を用意しているので、Eclipse で開発する場合は、あらかじめ「Preferences」の[Android]-[Build]タブの Custom debug keystore にファイルパスを設定しておくと、自社限定系のサンプルコードの動作確認に便利です。また、複数の APK から成るサンプルコードにおいて、各 APK 間の連携動作を確認するためには、各々の AndroidManifest.xml 内の android:debuggable の設定を合わせる必要があります。Eclipse から APK をインストールする場合は、明示的に設定が無ければ自動的に android:debuggable=”true”になります。

 All rights reserved © Japan Smartphone Security Association.

サンプルコードおよびキーストアファイルを Eclipse に取り込む方法については「2.5 サンプルコードの Eclipse への取り込み手順」をご参照ください。

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。

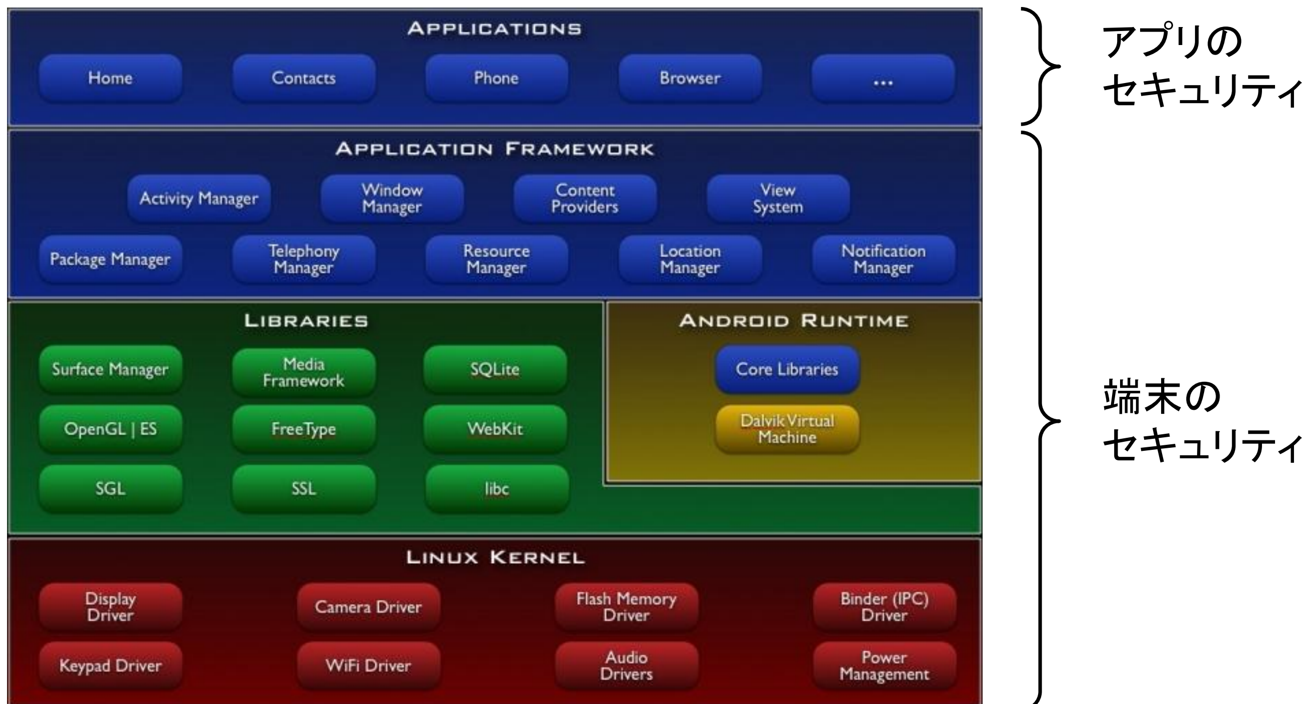
開発者のみなさんは常に多忙です。開発者の多くは、Android の深遠なるセキュリティの構造について深く理解することよりも、ある程度の Android セキュリティの知識を持って、迅速にかつ安全な Android アプリケーションをどんどん生産することが求められます。一方、セキュリティ設計が重要なアプリケーションもあります。このようなアプリケーションの開発者は Android のセキュリティについて深く理解している必要があります。

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。

All rights reserved © Japan Smartphone Security Association.

2.3. ガイド文書のスコープ

このガイド文書は一般の Android アプリケーション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 アプリケーション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 アプリケーションと、Android 端末メーカーがプレインストールする Android アプリケーション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、特に現行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。現行版では 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。なお、資産と脅威の扱いについては「3.1.3 資産分類と保護施策」にて詳しく説明しておりますので、合わせてご確認ください。

 All rights reserved © Japan Smartphone Security Association.

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/

2.5. サンプルコードの Eclipse への取り込み手順

サンプルコードの Eclipse への取り込み手順を説明します。サンプルコードは目的ごとに複数のプロジェクトにわかれています。これらのプロジェクトをまとめて取り込む方法を「2.5.1 サンプルプロジェクトを取り込む」に、選択して取り込む方法を「2.5.2 サンプルの各プロジェクトを選択して取り込む」に示します。プロジェクトの取り込みが終わったら「2.5.3 サンプルコード動作確認用 debug.keystore を設定する」を参照して debug.keystore ファイルを Eclipse に設定してください。なお、確認は下記の環境で行っております。

- OS
  - Windows 7 Ultimate SP1
- Eclipse
  - 4.2.0(Juno)
- Android SDK
  - Android 2.2(API 8)
    - 特に注意のないサンプルプロジェクトは Android 2.2(API 8)でビルドできます。但し、サンプルによっては、Android 2.3.3(API 10)以降の SDK Platform が必要な場合があります。

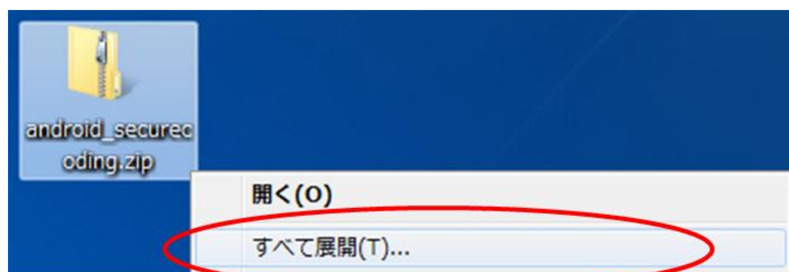
2.5.1. サンプルプロジェクトを取り込む

1. サンプルコードをダウンロードする

「2.2.1 サンプルコード」で紹介した URL よりサンプルコードを取得ます。

2. サンプルコードを展開する

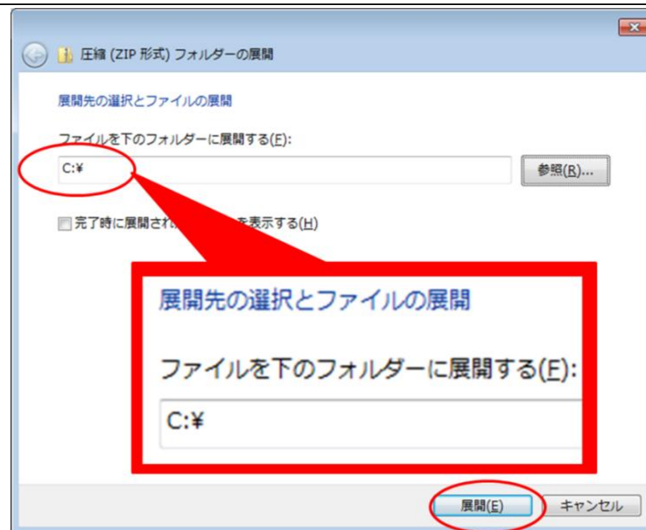
Zip で圧縮されたサンプルコードを右クリックし、表示されたメニューの“すべて展開”をクリックします。



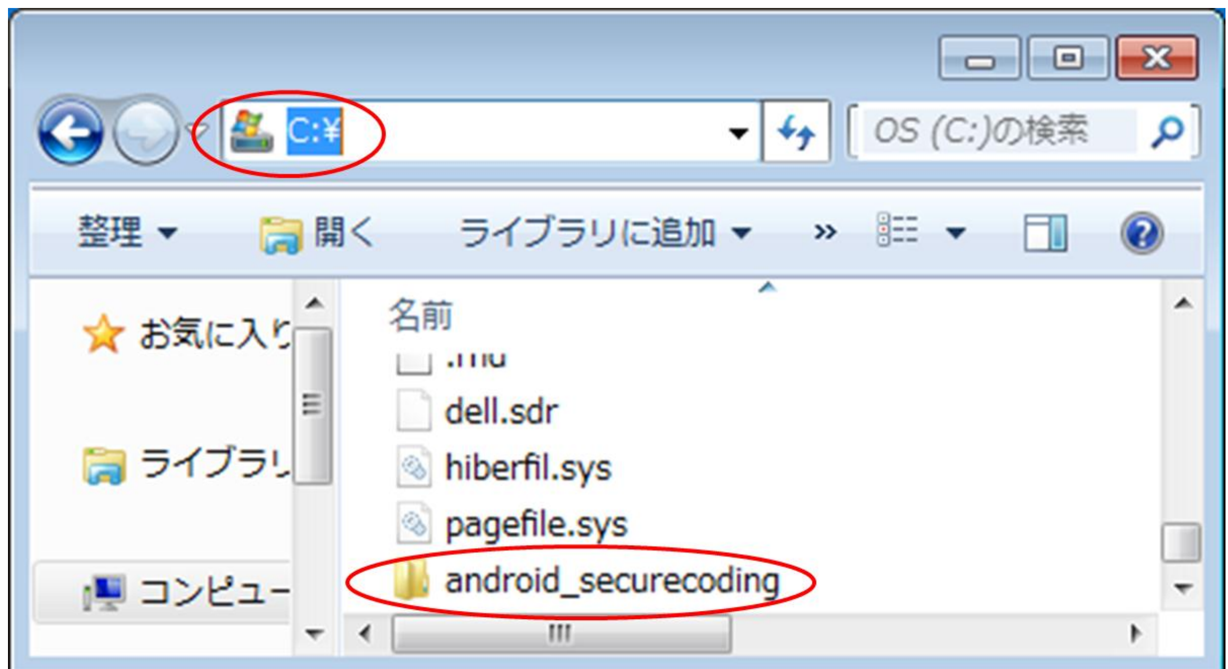
3. 展開先を指定する

ここでは“C:¥android_securecoding”という名前でワークスペースを作成します。そのため、"C:¥"を指定し"展開"ボタンをクリックします。

All rights reserved © Japan Smartphone Security Association.



"展開"ボタンをクリックすると“C:¥ ”直下に“android_securecoding”というフォルダが作成されます。



“android_securecoding”フォルダの中にはサンプルコードが含まれています。
例えば、「4.1 Activity を作る・利用する」の「4.1.1.3　パートナー限定 Activity を作る・利用する」においてサンプルコードを参照したい場合は以下をご覧ください。

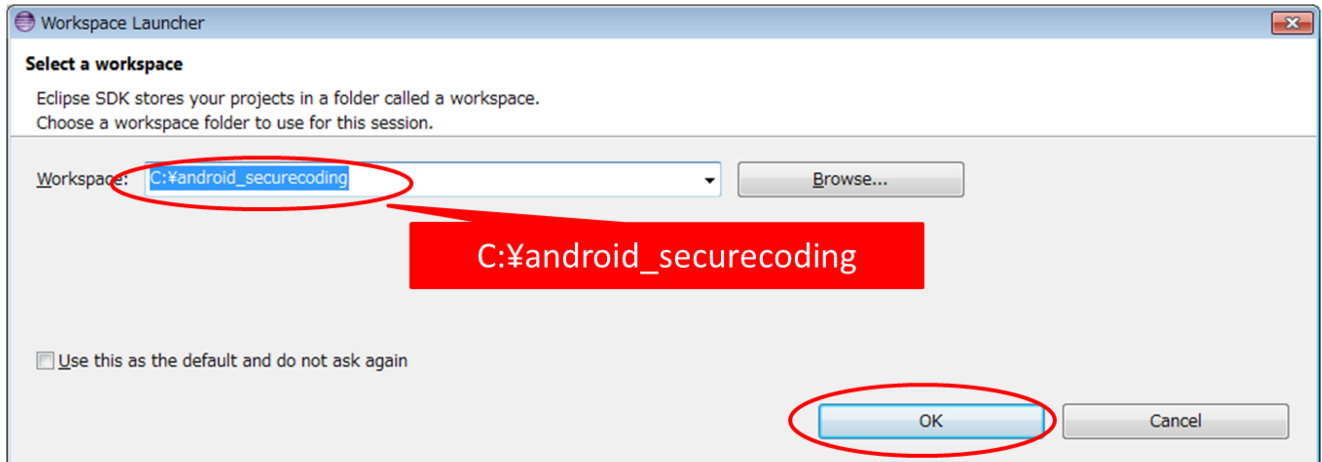
android_securecoding
　└Activity を作る・利用する
　　　└Activity ExclusiveActivity

以上のように、“android_securecoding”フォルダ配下は、「節タイトル」の直下に「サンプルコードのプロジェクト」が配置された構成となります。

All rights reserved © Japan Smartphone Security Association.

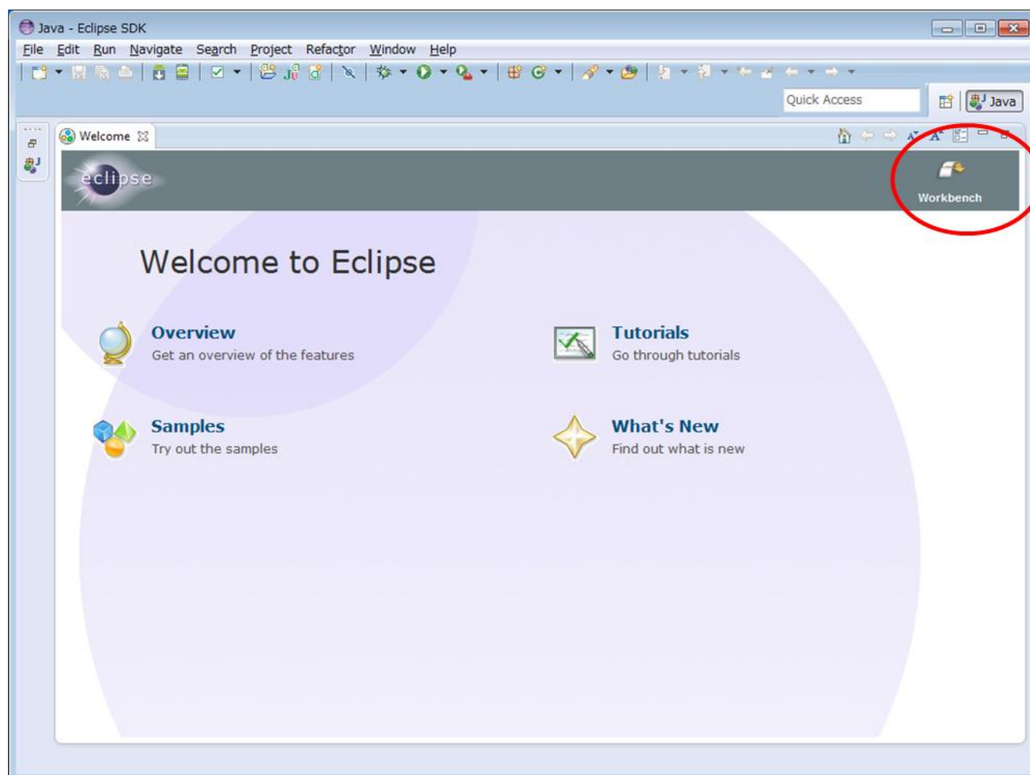
4. Eclipse を起動しワークスペースを指定する

スタートメニューやデスクトップアイコンなどから Eclipse を起動します。表示された選択ダイアログで、ワークスペースに、先ほどの手順にて展開した “C:¥android_securecoding”を指定します。選択ダイアログが表示されない場合はメニューより“File->Switch Workspace->Other…”をクリックしてください。



5. ワークベンチを表示する

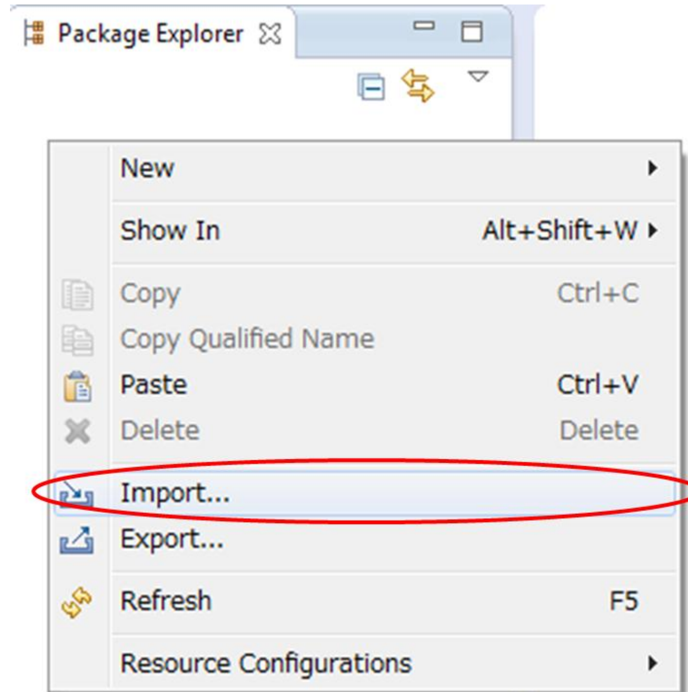
Eclipse が立ち上がったら、"ワークベンチ"をクリックします。



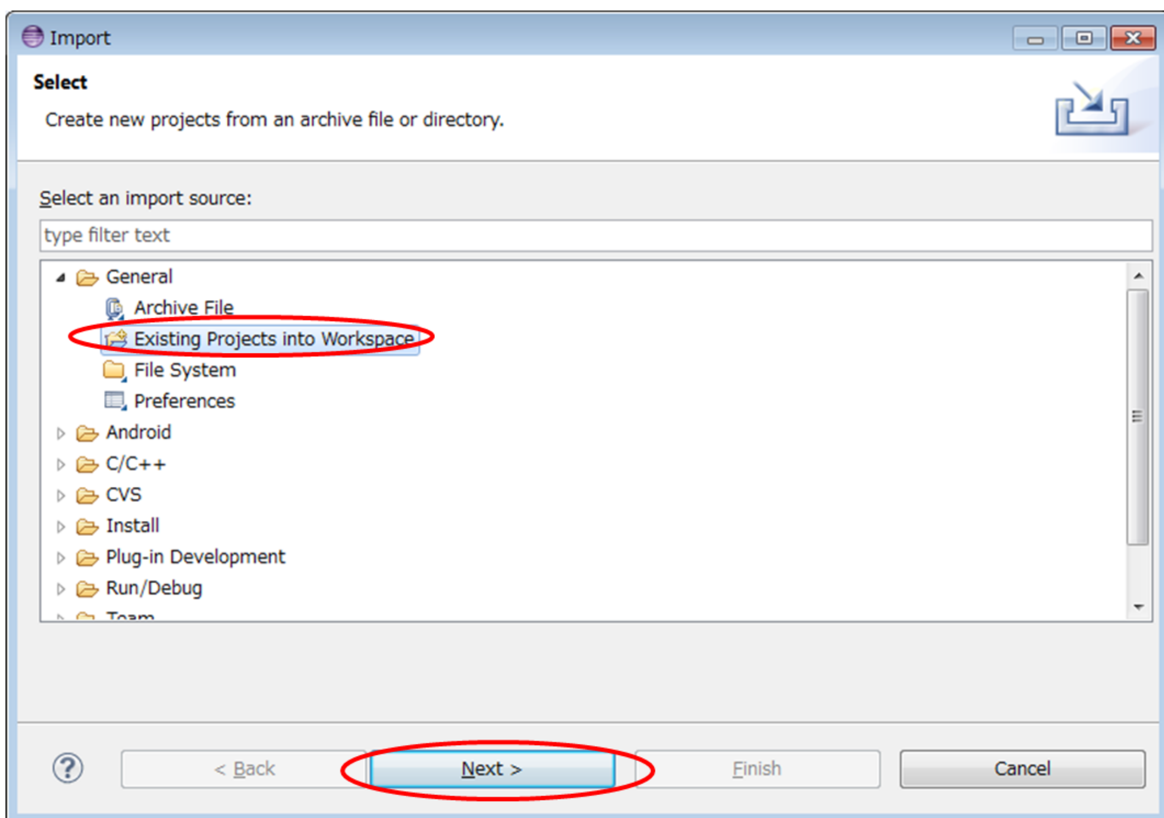
 All rights reserved © Japan Smartphone Security Association.

6. インポートを開始する

パッケージエクスプローラーで右クリックし、表示されたメニューの"Import"をクリックします。



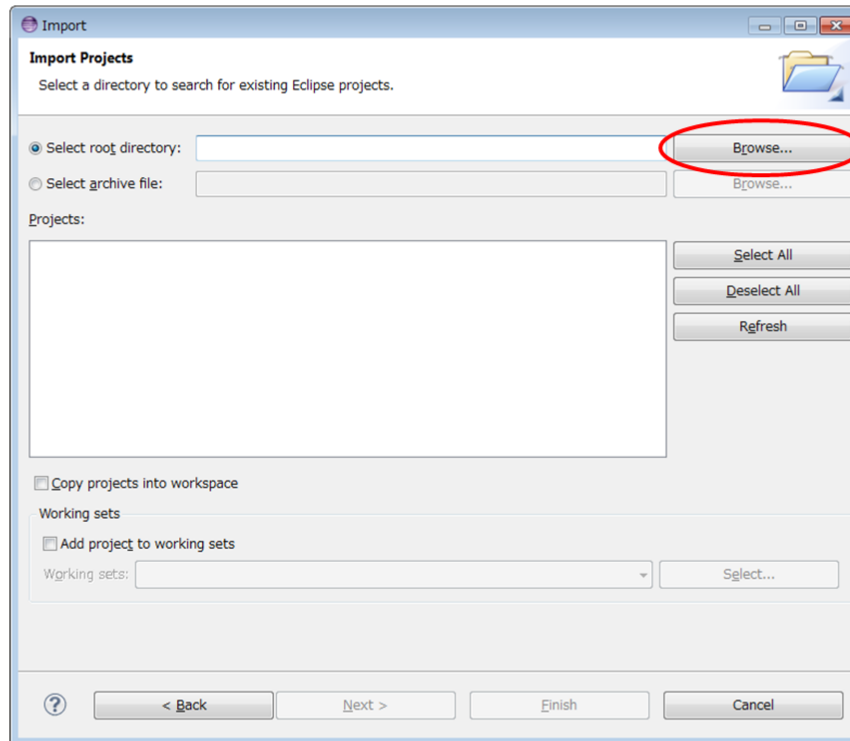
7. 既存プロジェクトをワークスペースにインポートする



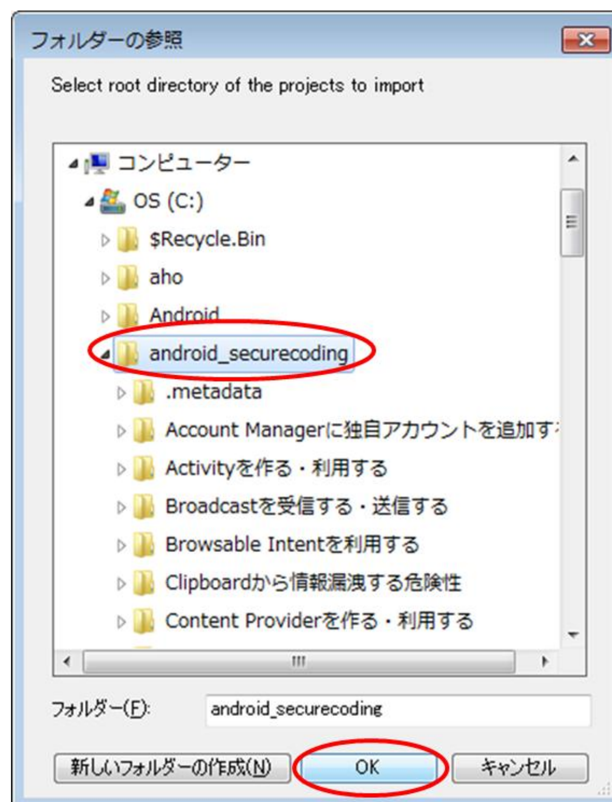
All rights reserved © Japan Smartphone Security Association.

8. プロジェクトを選択する

"Browse"をクリックしフォルダ参照画面を表示させます。



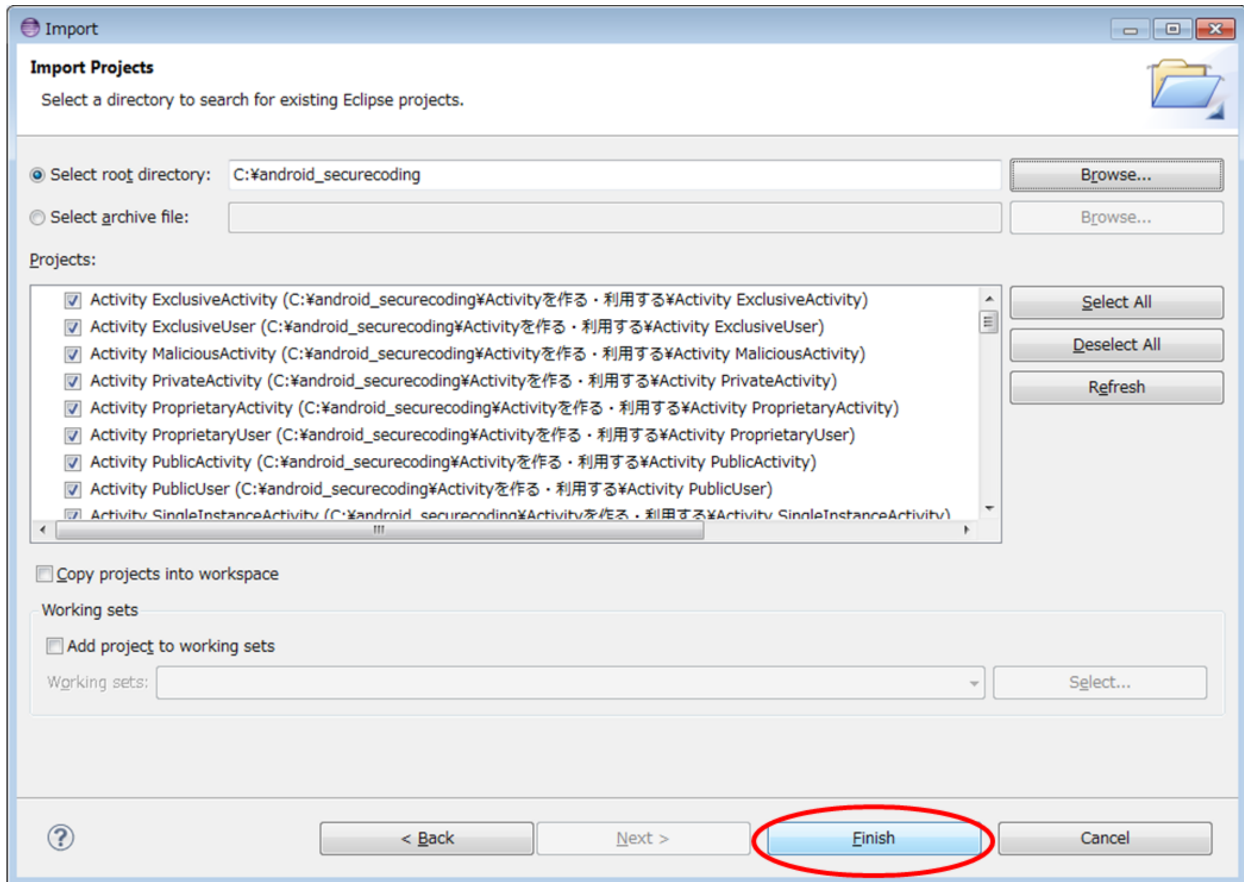
9. 展開したフォルダを選択する



 All rights reserved © Japan Smartphone Security Association.

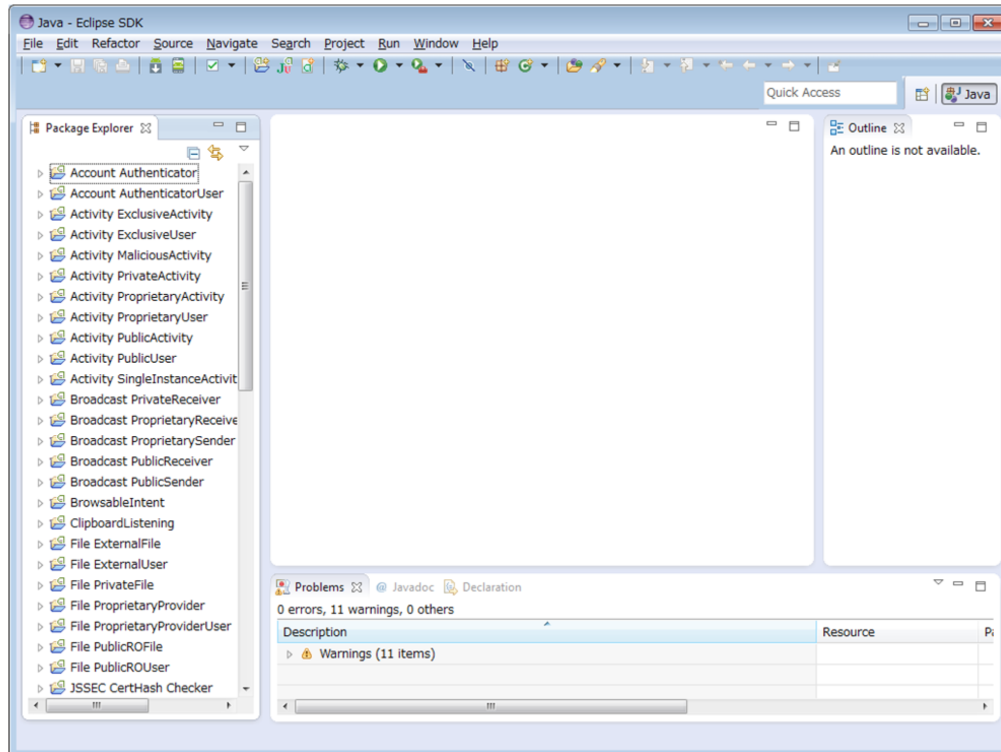
10. インポートを完了する

"Finish"を押下すると全プロジェクトがインポートされ完了します。
"Copy project into workspace"で取り込んでしまうと、各プロジェクトのフォルダ階層がフラットになってしまいます。プロジェクトによっては共通/JSSEC Shared を参照するものがあり、フォルダ階層が変わることでコンパイルエラーになってしまいます。ご注意ください。
一部のプロジェクトのみを取り込みたい場合は「2.5.2 サンプルの各プロジェクトを選択して取り込む」を参照してください。



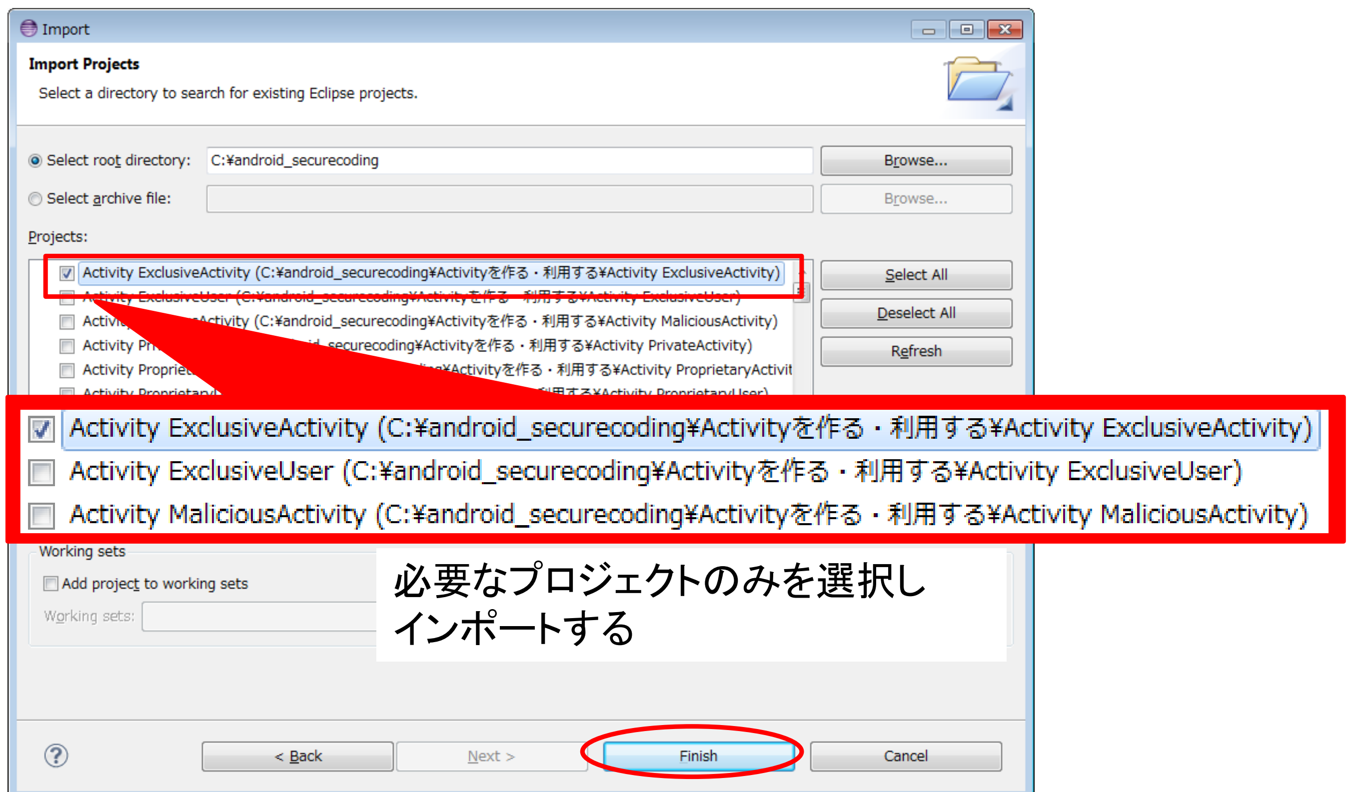
All rights reserved © Japan Smartphone Security Association.

11. 全プロジェクトがインポートされます



2.5.2. サンプルの各プロジェクトを選択して取り込む

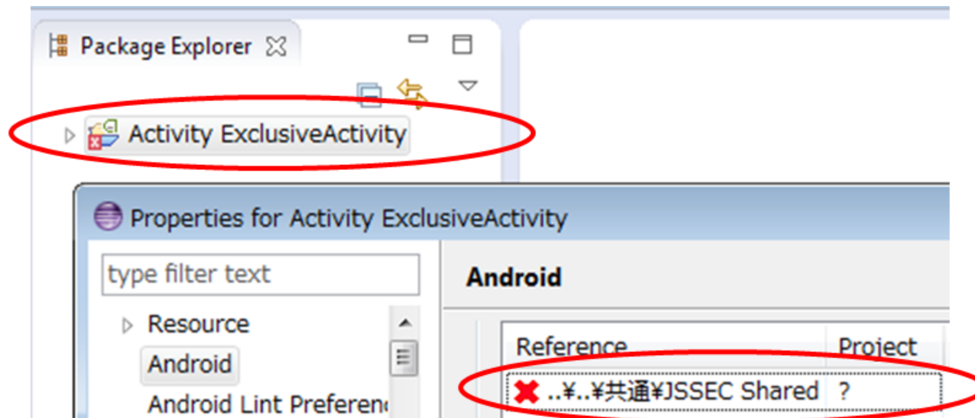
プロジェクトを選択してインポートします。Eclipse を起動し取り込む直前までは「2.5.1 サンプルプロジェクトを取り込む」を参照してください。



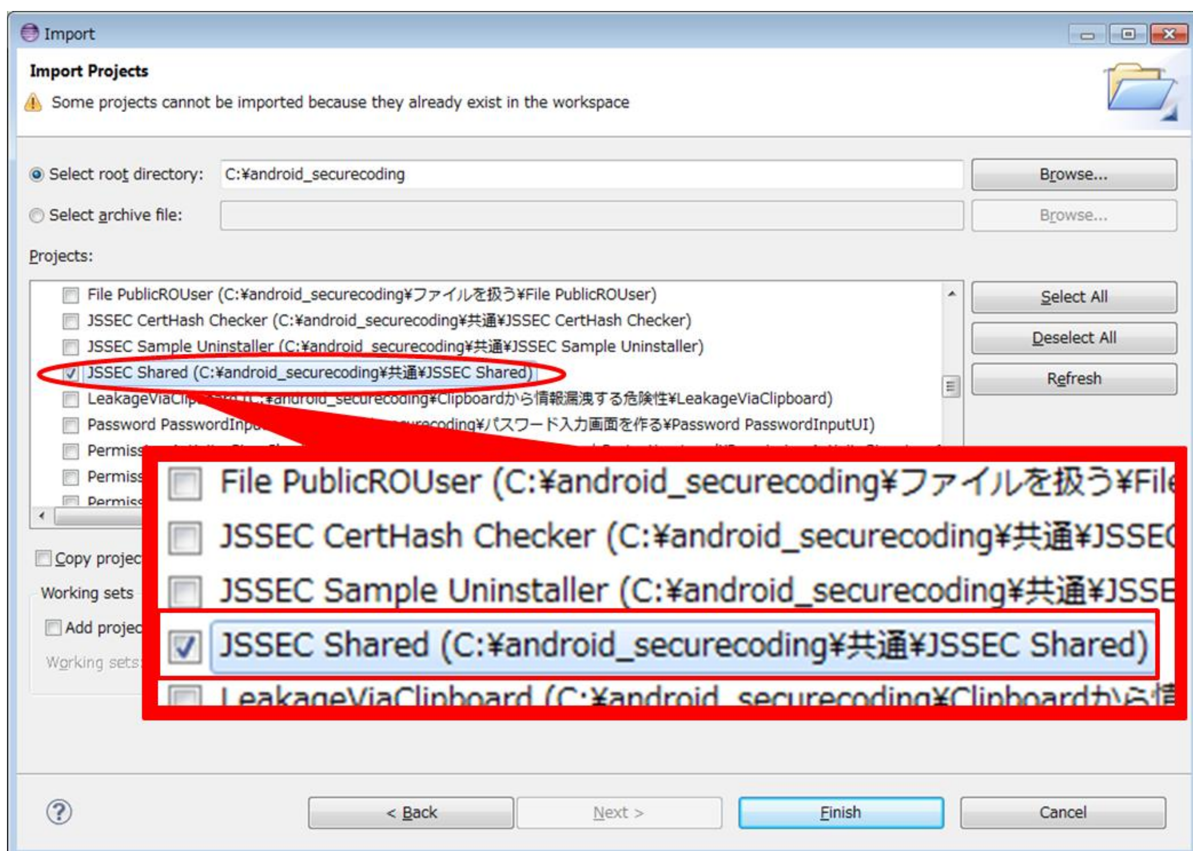
 All rights reserved © Japan Smartphone Security Association.

プロジェクトによっては"JSSEC Shared"を参照しているものもあります。

その場合、"JSSEC Shared"がインポートされていないと以下のようになります。必要に応じて"JSSEC Shared"をインポートしてください。



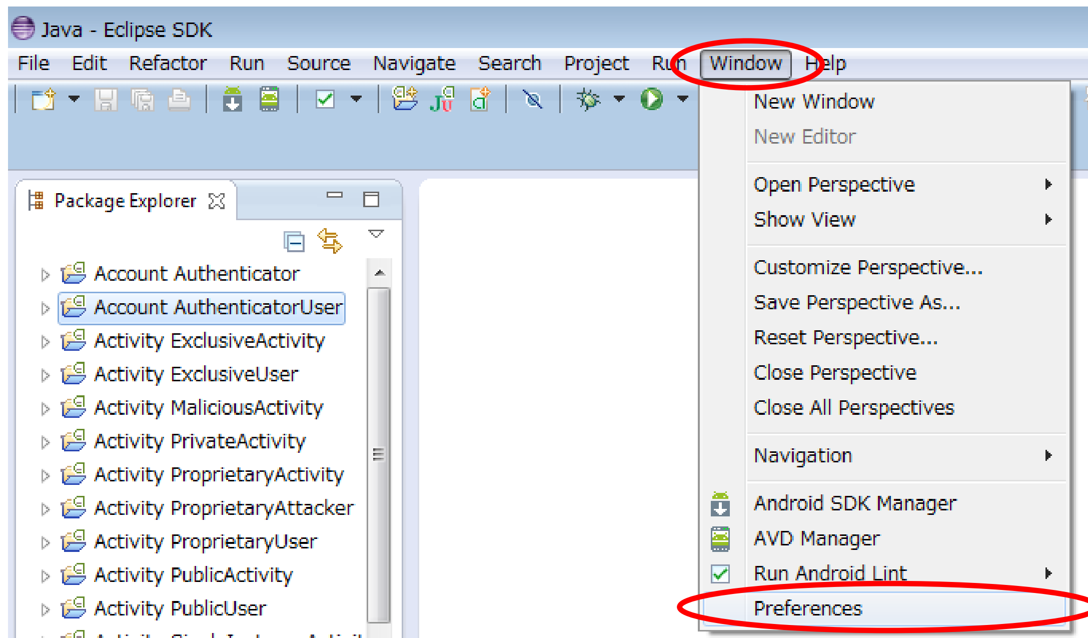
以下のように"JSSEC Shared"を選択してインポートします。



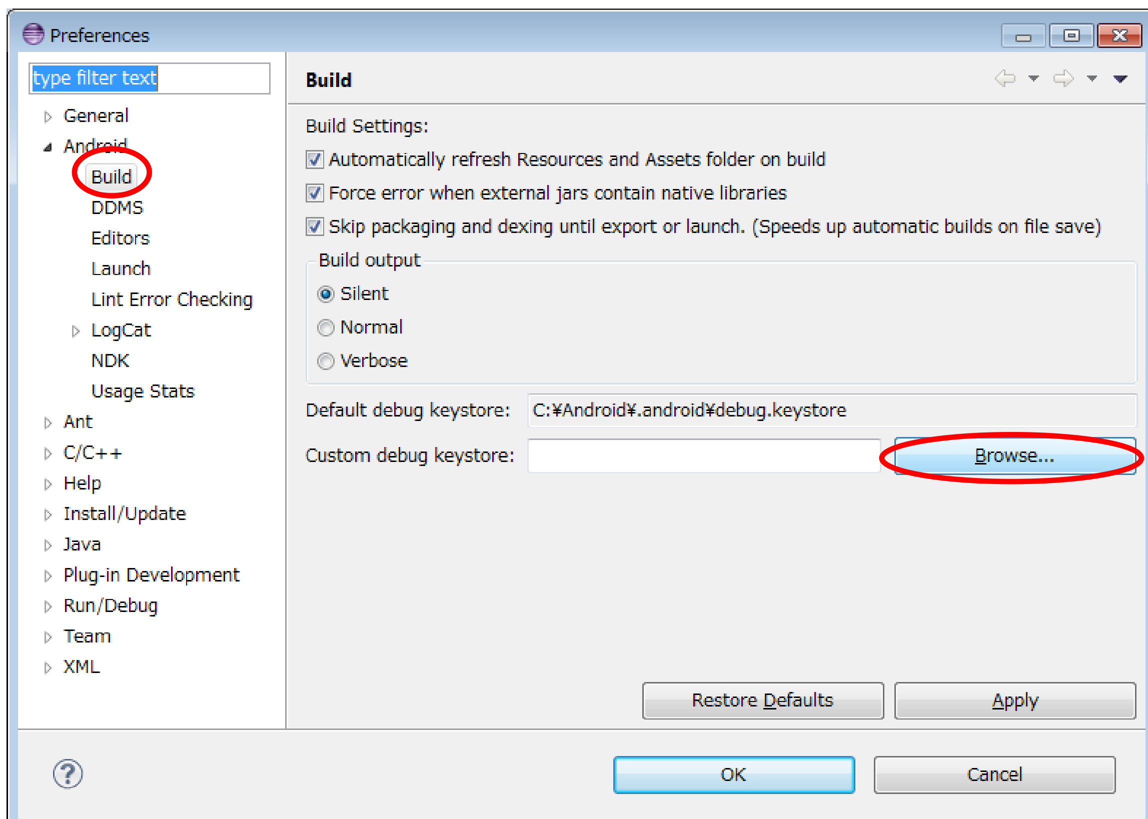
2.5.3. サンプルコード動作確認用 debug.keystore を設定する

サンプルコードから作成したアプリをを Android 端末やエミュレーターで動作させるためには署名が必要です。この署名に使うデバッグ用の鍵ファイル"debug.keystore"を Eclipse に設定します。

1. Window->Preferences をクリックする



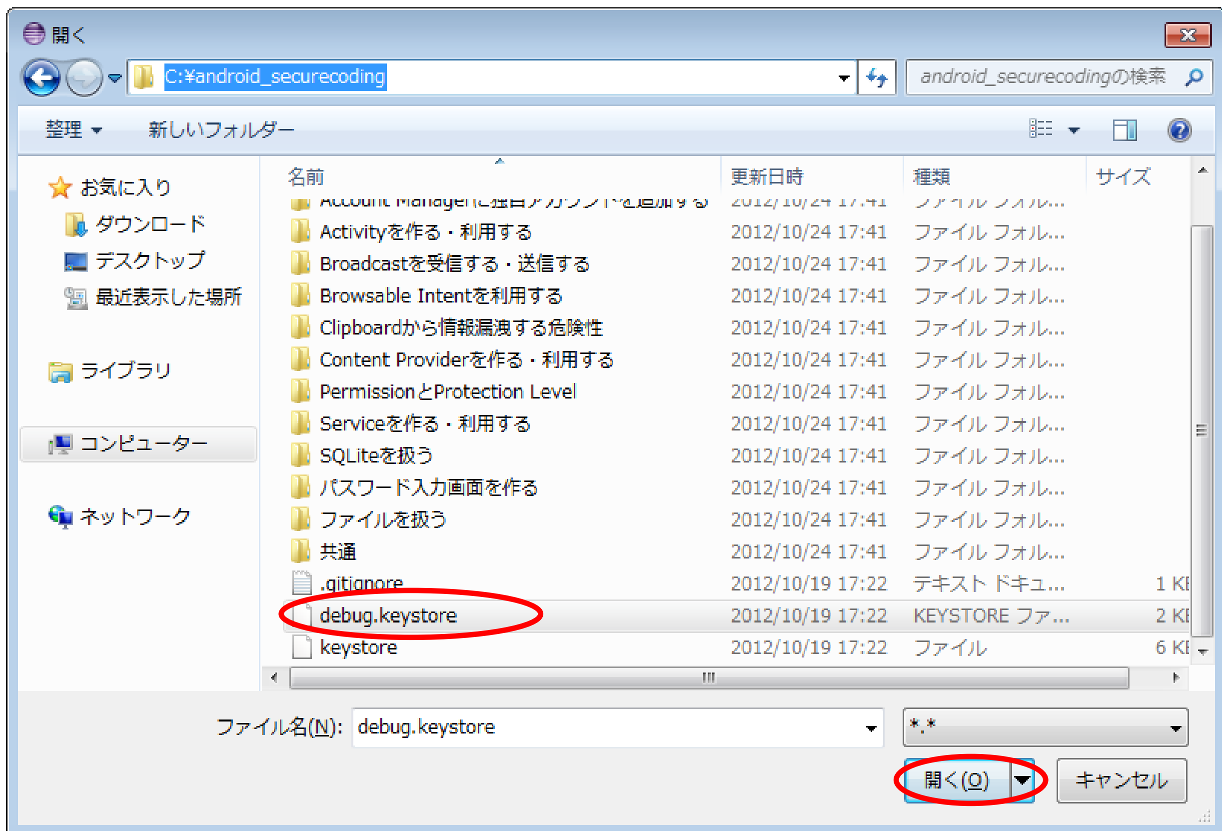
2. Android->Build を選択後 Browse...をクリックする



 All rights reserved © Japan Smartphone Security Association.

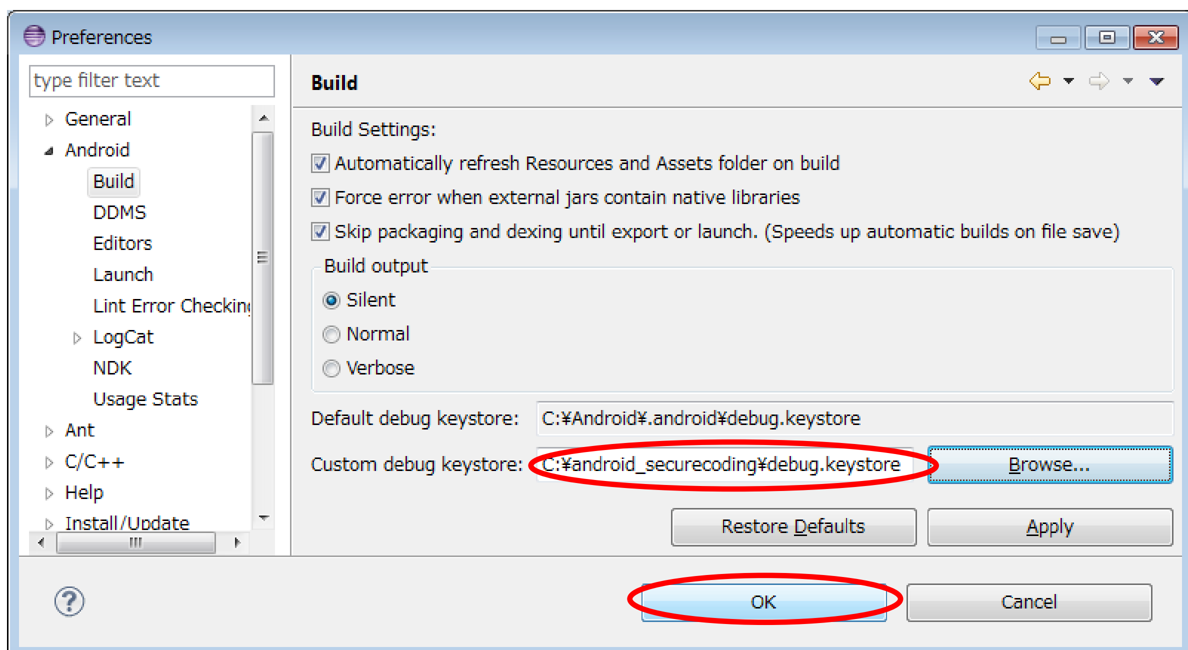
3. “debug.keystore”をクリックし開く

debug.keystore はサンプルコードに含まれています。(android_securecoding フォルダ直下)



4. OK をクリックし設定を適用する

最後に、下図のように、選択した debug.keystore のパスが表示されていることが確認できたら、OK をクリックします。



All rights reserved © Japan Smartphone Security Association.

3. セキュア設計・セキュアコーディングの基礎知識

このガイド文書は Android アプリ開発におけるセキュリティ Tips をまとめるものであるが、この章では Android スマートフォン／タブレットを例に一般的なセキュア設計・セキュアコーディングの基礎知識を扱う。後続の章において一般的なセキュア設計・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、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. Android アプリのセキュリティ

システムやアプリのセキュリティについて検討するとき、定番の考え方のフレームワークがある。まずそのシステムやアプリにおいて守るべき対象を把握する。これを「資産」と呼ぶ。次にその資産を脅かす攻撃を把握する。これを「脅威」と呼ぶ。最後に「資産」を「脅威」から守るための施策を検討・実施する。この施策を「対策」と呼ぶ。

ここで「対策」とは、システムやアプリに適切なセキュア設計・セキュアコーディングを施すことであり、このガイド文書では 4 章以降でこれを扱っている。本節では「資産」および「脅威」について焦点を当てる。

3.1.1. 「資産」 守るべき対象

システムやアプリにおける「守るべき対象」には「情報」と「機能」の 2 つがある。これらをそれぞれ「情報資産」と「機能資産」と呼ぶ。「情報資産」とは、許可された人だけが参照や変更ができる情報のことであり、それ以外の人には一切参照や変更ができてはならない情報のことである。「機能資産」とは許可された人だけが利用できる機能のことであり、それ以外の人には一切利用できてはならない機能のことである。

以下、Android スマートフォン／タブレットにおける情報資産と機能資産にどのようなものがあるかを紹介する。Android アプリや Android スマートフォン／タブレットを活用したシステムを開発するときの資産の洗い出しの参考にしてほしい。以降では、Android スマートフォン／タブレットを総称して Android スマートフォンと呼ぶ。

3.1.1.1. Android スマートフォンにおける情報資産

表 3.1.1-1 および表 3.1.1-2 は Android スマートフォンに入っている情報の一例である。これらの情報はスマートフォンユーザーに関する個人情報、プライバシー情報またはそれらに類する情報に該当するため適切な保護が必要である。

表 3.1.1-1 Android スマートフォンが管理する情報の例

情報	備考
電話番号	スマートフォン自身の電話番号
通話履歴	受発信の日時や相手番号
IMEI	スマートフォンの端末 ID
IMSI	回線契約者 ID

 All rights reserved © Japan Smartphone Security Association.

センサー情報	GPS、地磁気、加速度…
各種設定情報	WiFi 設定値…
アカウント情報	各種アカウント情報、認証情報…
メディアデータ	写真、動画、音楽、録音…
…	

表 3.1.1-2 アプリが管理する情報の例

情報	備考
電話帳	知人の連絡先
E メールアドレス	ユーザーのメールアドレス
E メールメールボックス	送受信メール本文、添付…
Web ブックマーク	ブックマーク
Web 閲覧履歴	閲覧履歴
カレンダー	予定、ToDo、イベント…
Facebook	SNS コンテンツ…
Twitter	SNS コンテンツ…
…	

表 3.1.1-1 の情報は主に Android スマートフォン本体または SD カードに含まれる情報であり、表 3.1.1-2 の情報は主にアプリが管理する情報である。特に表 3.1.1-2 の情報については、アプリがインストールされればされるほど、どんどん本体の中に増えていくことになるのである。

表 3.1.1-3 は電話帳の 1 件のエントリに含まれる情報である。この情報はスマートフォンユーザーに関する情報ではなく、スマートフォンユーザーの知人、友人等に関する情報である。つまりスマートフォンにはその利用者であるユーザーのみならず、ほかの人々の情報も含まれていることに注意が必要だ。

表 3.1.1-3 電話帳(Contacts)の 1 件のエントリに含まれる情報の例

情報	内容
電話番号	自宅、携帯電話、仕事、FAX、MMS…
E メールアドレス	自宅、仕事、携帯電話…
プロフィール画像	サムネール画像、大きな画像…
インスタントメッセンジャー	AIM、MSN、Yahoo、Skype、QQ、Google Talk、ICQ、Jabber、Netmeeting…
ニックネーム	略称、イニシャル、旧姓、別名…
住所	国、郵便番号、地域、地方、町、通り…
グループ	お気に入り、家族、友達、同僚…
ウェブサイト	ブログ、プロフィールサイト、ホームページ、FTP サーバー、自宅、会社…
イベント	誕生日、記念日、その他…
関係する人物	配偶者、子供、父、母、マネージャー、助手、同棲関係、パートナー…

All rights reserved © Japan Smartphone Security Association.

SIP アドレス	自宅、仕事、その他…
…	…

これまでの説明では主にスマートフォンユーザーの情報を紹介してきたが、アプリはユーザー以外の情報も扱っている。図 3.1-1 は 1 つのアプリが管理している情報を表しており、大きく分けるとプログラム部分とデータ部分に分かれる。プログラム部分は主にアプリメーカーの情報であり、データ部分は主にユーザーの情報である。アプリメーカーの情報の中には、勝手にユーザーに利用されたくない情報もあり得るため、そうした情報についてはユーザーが参照・変更できないような保護が必要である。

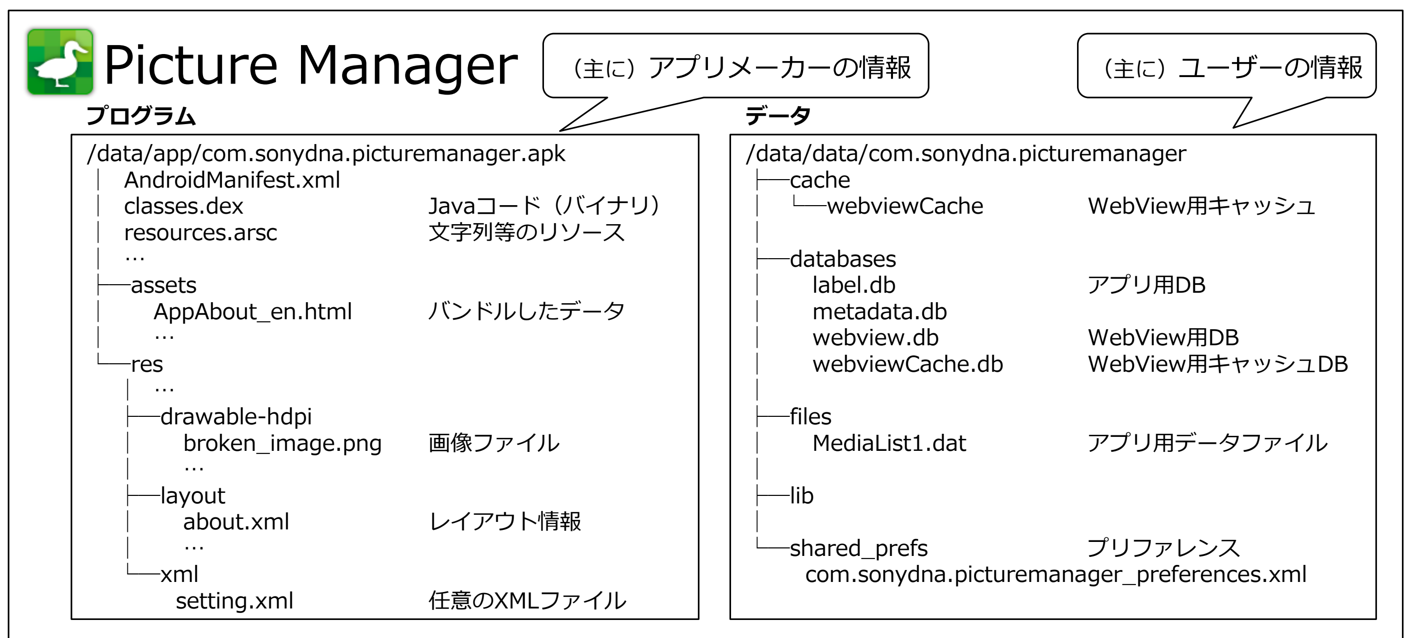


図 3.1-1 アプリが抱えている情報

Android アプリを作る場合には図 3.1-1 のようなアプリ自身が管理する情報のみならず、表 3.1.1-1、表 3.1.1-2、表 3.1.1-3 のような Android スマートフォン本体や他のアプリから取得した情報に関しても適切に保護する必要があることにも注意が必要だ。

3.1.1.2. Android スマートフォンにおける機能資産

表 3.1.1-4 は Android OS がアプリに提供する機能の一例である。これらの機能がマルウェア等に勝手に利用されてしまうとユーザーの意図しない課金が生じたり、プライバシーが損なわれるなどの被害が生じたりする。そのため情報資産と同様にこうした機能資産も適切に保護されなければならない。

表 3.1.1-4 Android OS がアプリに提供する機能 の一例

機能	機能
SMS メッセージを送受する機能	カメラ撮影機能
電話を掛ける機能	音量変更機能
ネットワーク通信機能	電話番号、携帯状態の読み取り機能

All rights reserved © Japan Smartphone Security Association.

GPS 等で現在位置を得る機能	SD カード書き込み機能
Bluetooth 通信機能	システム設定変更機能
NFC 通信機能	ログデータの読み取り機能
インターネット通話（SIP）機能	実行中アプリ情報の取得機能
…	…

Android OS がアプリに提供する機能に加え、Android アプリのアプリ間連携機能も機能資産に含まれる。Android アプリはそのアプリ内で実現している機能を他のアプリから利用できるように提供することができ、このような仕組みをアプリ間連携と呼んでいる。この機能は便利である反面、Android アプリの開発者がセキュアコーディングの知識がないために、アプリ内部だけで利用する機能を誤って他のアプリから利用できるようにしてしまっているケースもある。他のアプリから利用できる機能の内容によっては、マルウェアから利用されては困ることもあるため、意図したアプリだけから利用できるように適切な保護が必要となることがある。

3.1.2. 「脅威」 資産を脅かす攻撃

前節では Android スマートフォンにおける資産について解説した。ここではそれらの脅威、つまり資産を脅かす攻撃について解説する。資産が脅かされるとは簡単に言えば、情報資産が他人に勝手に参照・変更・削除・作成されることを言い、機能資産が他人に勝手に利用されることを言う、といった具合だ。こうした資産を直接的および間接的に操作する攻撃行為を「脅威」と呼ぶ。また攻撃行為を行う人や物のことを「脅威源」と呼ぶ。攻撃者やマルウェアは脅威源であって脅威ではない。攻撃者やマルウェアが行う攻撃行為のことを脅威と呼ぶのである。これら用語間の関係を図 3.1-2 資産、脅威、脅威源、脆弱性、被害の関係に示す。

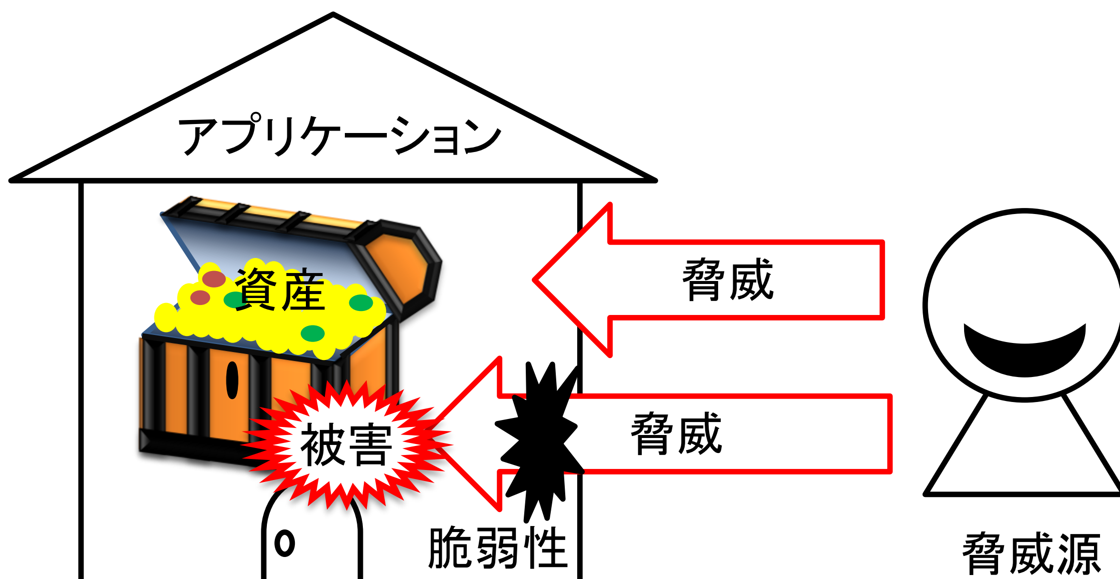


図 3.1-2 資産、脅威、脅威源、脆弱性、被害の関係

図 3.1-3 は Android アプリが動作する一般的な環境を表現したものだ。以降ではこの図をベースにして Android アプリにおける脅威の説明を展開するため、初めにこの図の見方を解説する。

All rights reserved © Japan Smartphone Security Association.

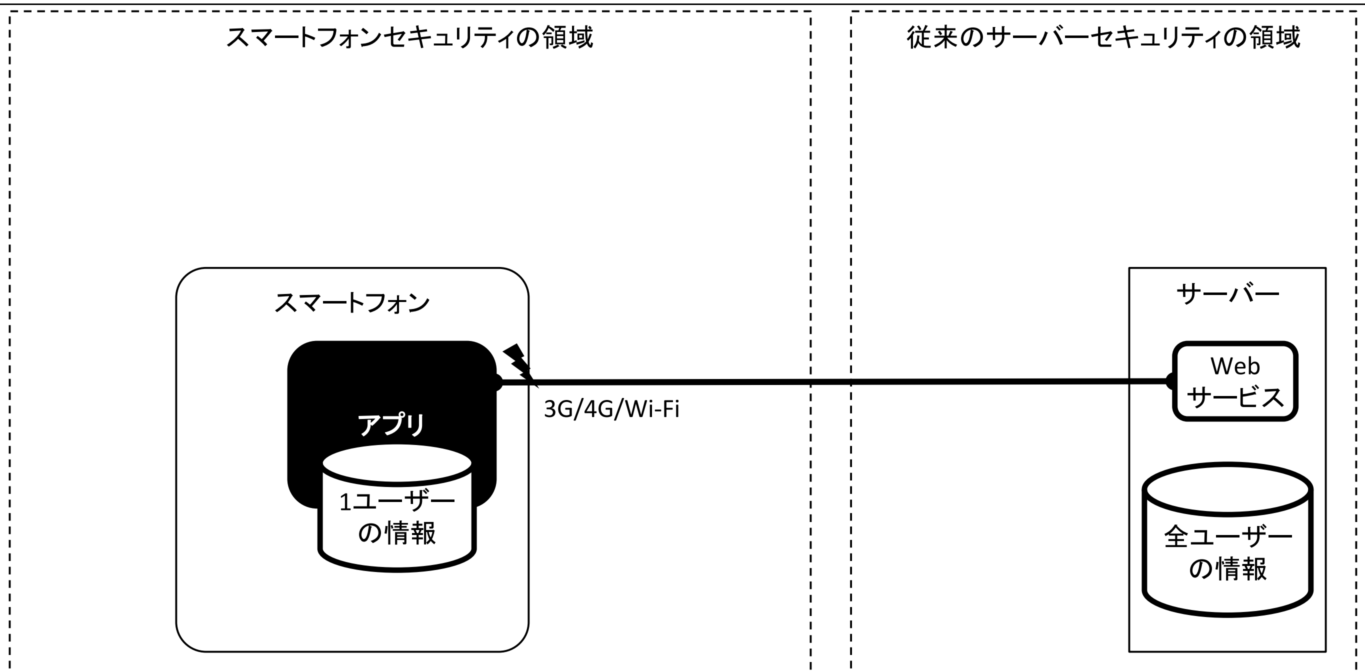


図 3.1-3 Android アプリが動作する一般的な環境

図の左右にスマートフォンとサーバーを配置している。スマートフォンやサーバーは 3G/4G/Wi-Fi およびインターネットを経由して通信している。スマートフォンの中には複数のアプリが存在するが、以降の説明で 1 つのアプリに関する脅威を説明するため、この図では 1 つのアプリに絞って説明している。スマートフォン上のアプリはそのユーザーの情報を主に扱うが、サーバー上の Web サービスは全ユーザーの情報を集中管理することを表現している。そのため従来同様にサーバーセキュリティの重要性は変わらない。サーバーセキュリティについては、このガイド文書ではスコープ外であるため言及しない。

以降ではこの図を使って Android アプリにおける脅威を説明していく。

All rights reserved © Japan Smartphone Security Association.

3.1.2.1. ネットワーク上の第三者による脅威

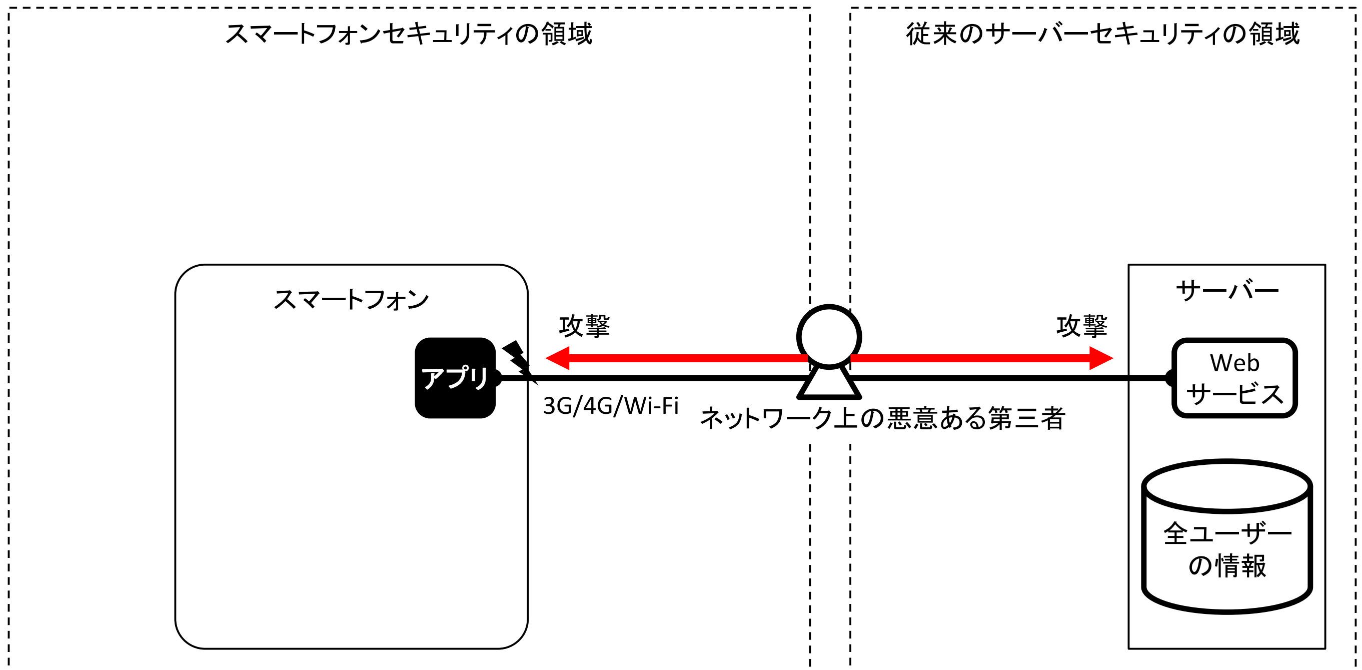


図 3.1-4 ネットワーク上の悪意ある第三者がアプリを攻撃する

スマートフォンアプリはユーザーの情報をサーバーで管理する形態が一般的である。そのため情報資産がネットワーク上を移動することになる。図 3.1-4 に示すように、ネットワーク上の悪意ある第三者は通信中の情報を参照(盗聴)したり、情報を変更(改ざん)したりしようとする。また本物のサーバーになりすまして、アプリの通信相手になろうとする。もちろん従来同様、ネットワーク上の悪意ある第三者はサーバーも攻撃する。

3.1.2.2. ユーザーがインストールしたマルウェアによる脅威

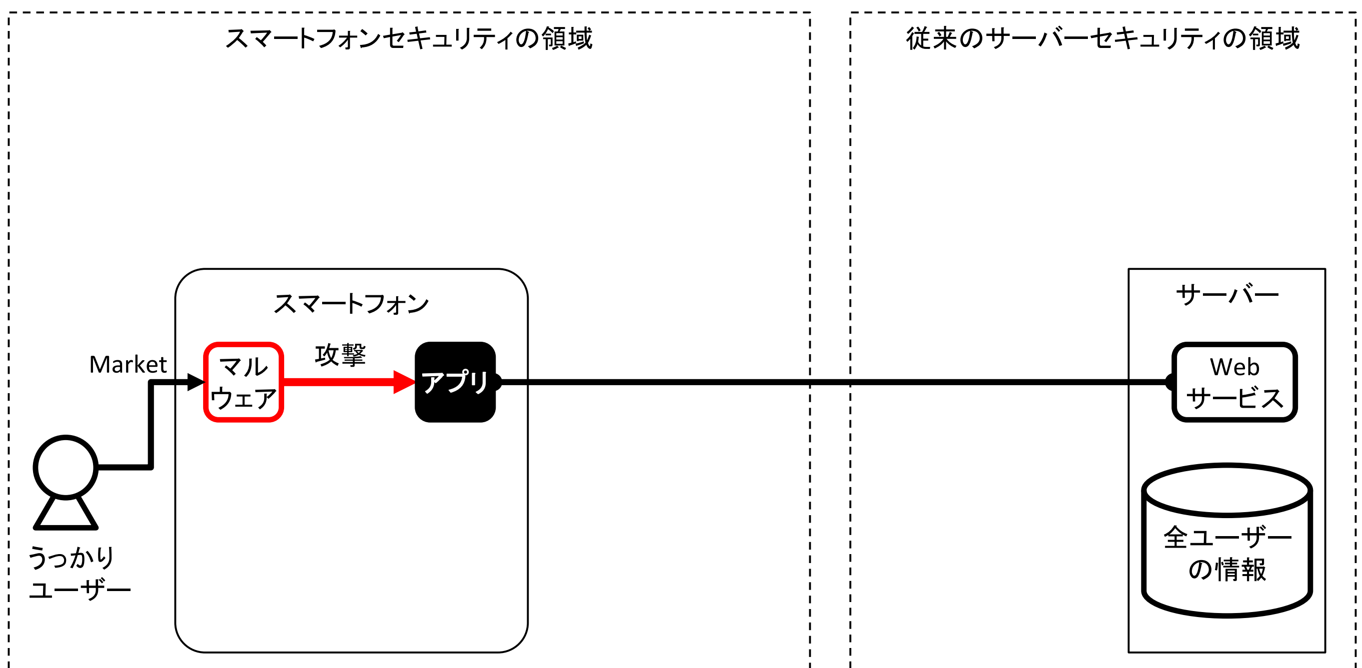


図 3.1-5 ユーザーがインストールしてしまったマルウェアがアプリを攻撃する

スマートフォンは多種多様なアプリをマーケットから入手しインストールすることで機能拡張できることがその最大の

All rights reserved © Japan Smartphone Security Association.

特徴である。ユーザーがうっかりマルウェアアプリをインストールしてしまうこともある。図 3.1-5 が示すように、マルウェアはアプリ間連携機能やアプリの脆弱性を悪用してアプリの情報資産や機能資産にアクセスしようとする。

3.1.2.3. アプリの脆弱性を悪用する攻撃ファイルによる脅威

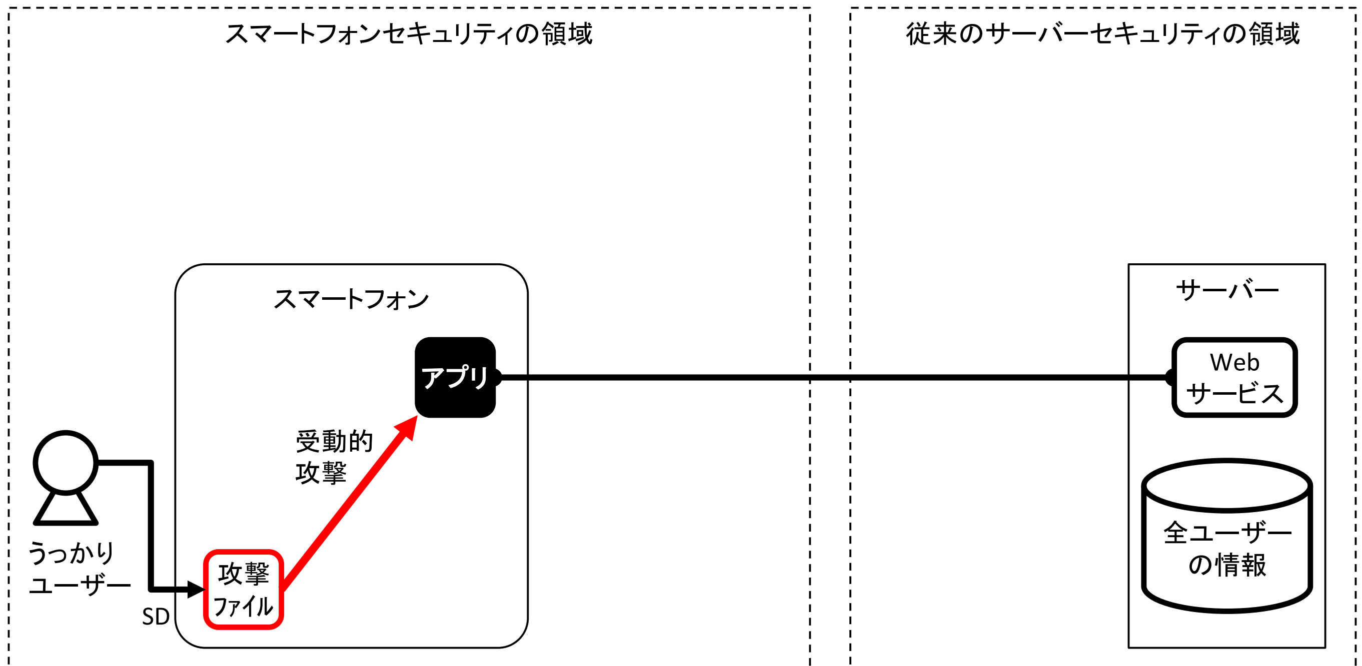


図 3.1-6 アプリの脆弱性を悪用する攻撃ファイルがアプリを攻撃する

インターネット上には音楽や写真、動画、文書など、様々なタイプのファイルが大量に公開されており、ユーザーがそれらのファイルを SD カードにダウンロードし、スマートフォンで利用する形態が一般的である。またスマートフォンで受信したメールに添付されるファイルを利用する形態も一般的である。これらのファイルは閲覧用や編集用のアプリでオープンされ利用される。

こうしたファイルを処理するアプリの機能に脆弱性があると、攻撃ファイルにより、そのアプリの情報資産や機能資産が悪用されてしまう。特に複雑なデータ構造を持ったファイル形式の処理においては脆弱性が入り込みやすい。攻撃ファイルは巧みに脆弱性を悪用してアプリを操作し、攻撃ファイルの作成者の目的を達成する。

図 3.1-6 に示すように、攻撃ファイルは脆弱なアプリによってオープンされるまでは何もせず、いったんオープンされるとアプリの脆弱性を悪用した攻撃を始める。攻撃者が能動的に行う攻撃行為と比較して、このような攻撃手法を受動的攻撃（Passive Attack）と呼ぶ。

 All rights reserved © Japan Smartphone Security Association.

3.1.2.4. 悪意あるスマートフォンユーザーによる脅威

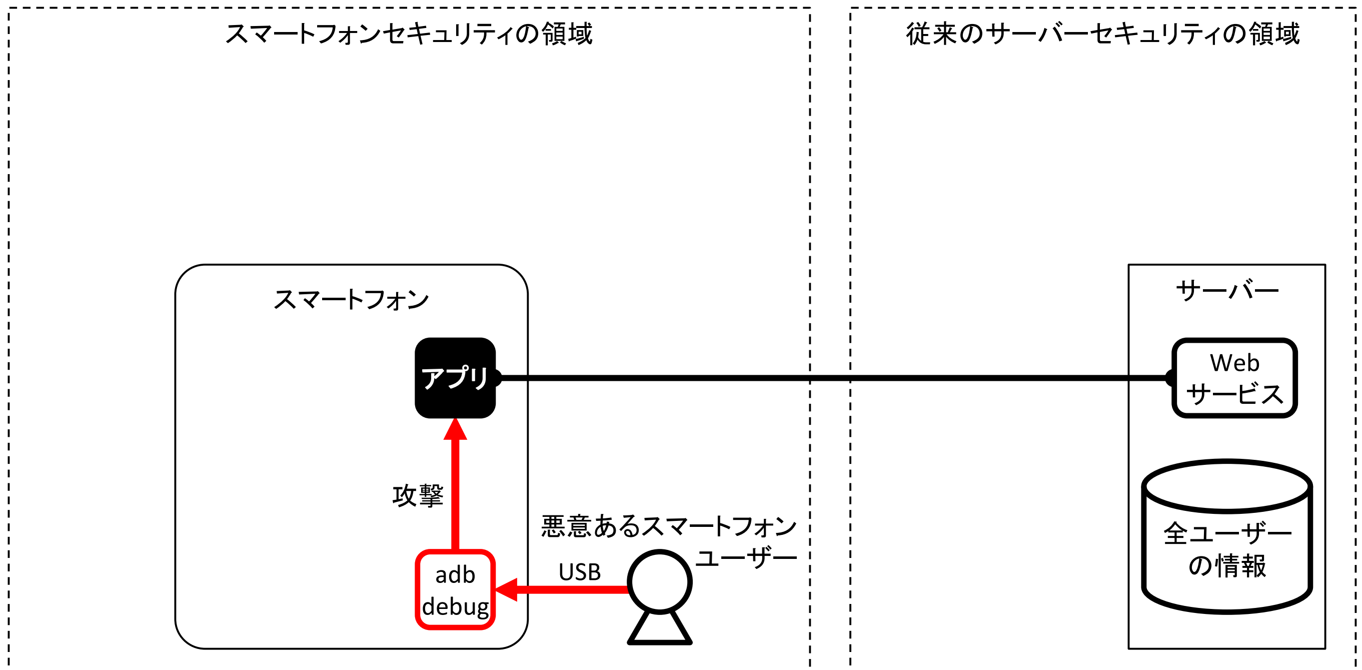


図 3.1-7 悪意あるスマートフォンユーザーがアプリを攻撃する

Android スマートフォンのアプリ開発においては、一般ユーザーに対してアプリを開発、解析する環境や機能が公式に提供されている。提供されている機能の中でも、特に ADB と呼ばれる充実したデバッグ機能は、誰でも何の登録・審査もなく利用可能であり、この機能により Android スマートフォンユーザーは OS 解析行為やアプリ解析行為を容易に行うことができる。

図 3.1-7 に示すように、悪意あるスマートフォンユーザーは ADB 等のデバッグ機能を利用してアプリを解析し、アプリが抱える情報資産や機能資産にアクセスしようとする。アプリが抱える資産がユーザー自身のものである場合には問題とならないが、アプリメーカー等ユーザー以外のステークホルダーの資産である場合に問題となる。このようにスマートフォンのユーザー自身が悪意を持ってアプリ内の資産を狙うことがあることにも注意が必要だ。

All rights reserved © Japan Smartphone Security Association.

3.1.2.5. スマートフォンの近くにいる第三者による脅威

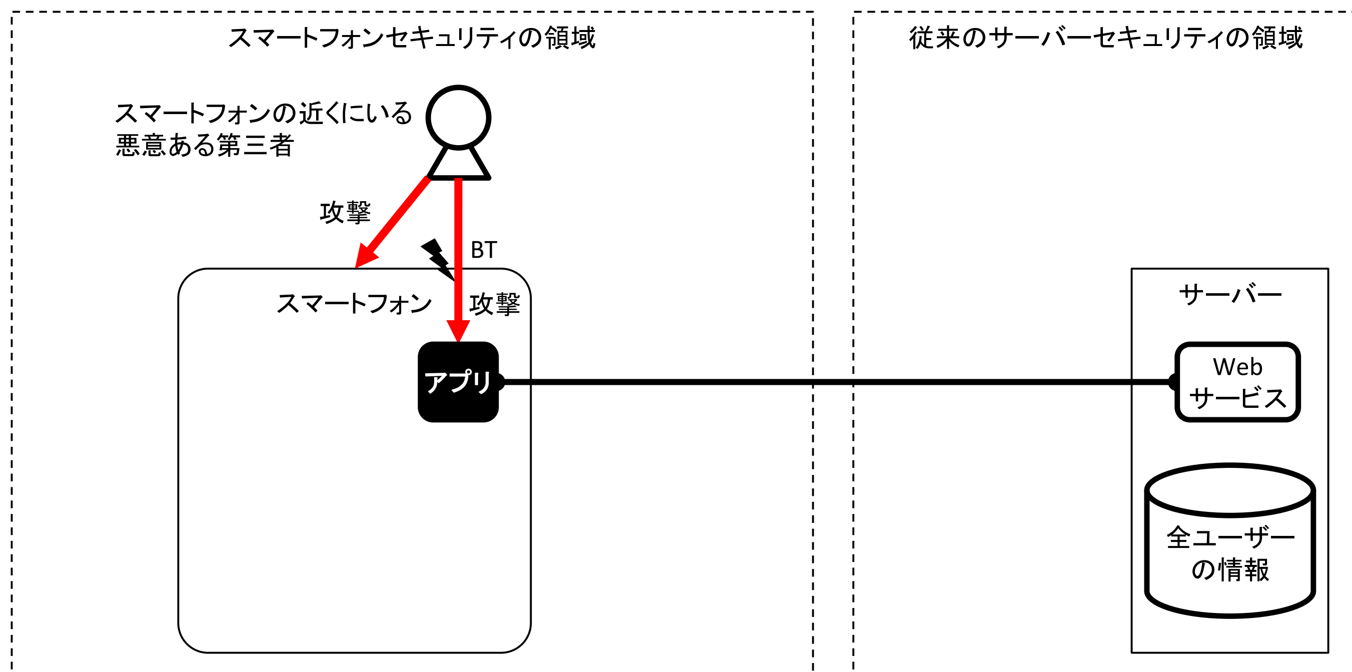


図 3.1-8 スマートフォンの近くにいる悪意ある第三者がアプリを攻撃する

スマートフォンはその携帯性の良さや、Bluetooth 等の近距離無線通信機能を標準搭載していることから、物理的にスマートフォンの近くにいる悪意ある第三者から攻撃され得ることも忘れてはならない。攻撃者はユーザーが入力中のパスワードを肩越しに覗き見するショルダーハックをしたり、図 3.1-8 に示すように、Bluetooth 通信機能を持つアプリに対して Bluetooth でアクセスしたり、スマートフォン自体を盗んだり破壊したりする。特にスマートフォン自体の窃盗や破壊については、機密度の高さからスマートフォン内から一切外に出さない運用としている情報資産が失われてしまう脅威であり、アプリ設計の段階で見過ごされてしまうこともあるので注意が必要だ。

 All rights reserved © Japan Smartphone Security Association.

3.1.2.6. 様々な脅威

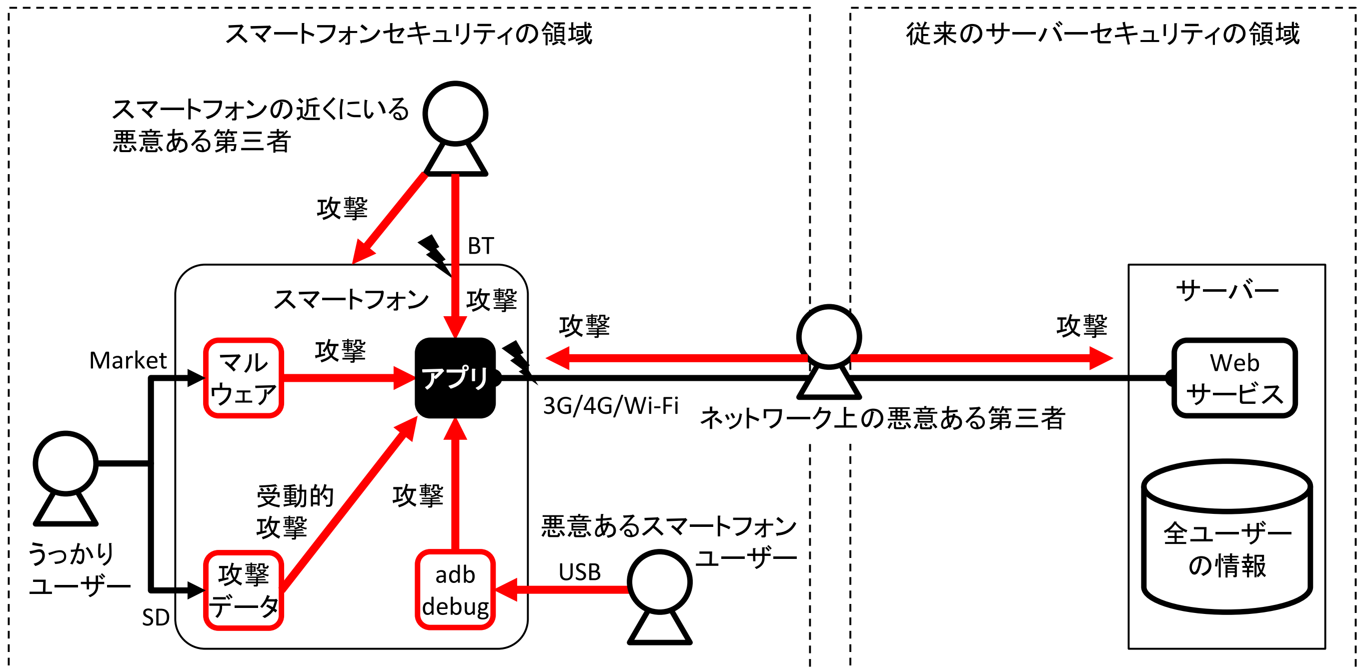


図 3.1-9 スマートフォンアプリは様々な攻撃にさらされている

図 3.1-9 はこれまで説明した脅威をひとまとめにした図である。このようにスマートフォンアプリを取り巻く脅威には様々なものがあり、この図はそのすべてを書き出しているものではない。日々の情報収集により、Android アプリを取り巻く脅威について認識を広め、アプリのセキュア設計・セキュアコーディングに活かしていく努力が必要である。日本スマートフォンセキュリティ協会（JSSEC）が作成した次の文書もスマートフォンの脅威について役立つ情報を提供しているので参考にしていただきたい。

- 『スマートフォン＆タブレットの業務利用に関するセキュリティガイドライン』【第一版】
  http://www.jssec.org/dl/guidelines2011_v1.0.pdf
  http://www.jssec.org/dl/guidelines2012Enew_v1.0.pdf (English)

- 『スマートフォンネットワークセキュリティ実装ガイド』【β版】
  http://www.jssec.org/dl/NetworkSecurityGuideB1.pdf

- 『スマートフォンの業務利用におけるクラウド活用ガイド』【β版】
  http://www.jssec.org/dl/cloudguide2012_beta.pdf

- 『MDM 導入・運用検討ガイド』【β版】
  http://www.jssec.org/dl/MDMGuideV1B.pdf

All rights reserved © Japan Smartphone Security Association.

3.1.3. 資産分類と保護施策

前節までで解説した通り Android スマートフォンには様々な脅威が存在する。それら脅威から、アプリが扱うすべての資産を保護することは、開発にかかる時間や技術的限界などから困難な場合がある。そのため、Android アプリ開発者は、アプリが扱う資産の重要度や容認される被害レベルを判断基準とした優先度に応じて、資産に対する妥当な対策を検討することが必要になる。

資産毎の保護施策を決めるため、アプリが扱うそれぞれの資産について、その重要度・保護に関する法的根拠・被害発生時の影響・開発者（ないし組織）の社会的責任などを検討した上で、資産を分類し、分類毎に保護施策のレベルを定める。これが、それぞれの資産をどのように扱い、どのような対策を施すかを決める判断基準となる。この分類はアプリ開発者（ないし組織）として資産をどのように扱い保護するかを定める基準そのものであるため、アプリ開発者（または組織）がそれぞれの事情に合わせて、分類方法や対策内容を定める必要がある。

参考までに、本ガイドにおける資産分類と保護施策のレベルを以下に示す。

表 3.1.3-1 資産分類と保護施策のレベル

資産分類	資産のレベル	保護施策のレベル
高位	資産が被害にあった場合、組織または個人の活動に致命的または壊滅的な影響をあたえるもの 例）資産が被害にあった場合、当該組織が事業を継続できなくなるレベル	• Android OS のセキュリティモデルを破る、root 権限を奪取した状態からの攻撃や APK の dex 部分を改造するといった、高度な攻撃に対しても保護する • UX 等の他要素よりもセキュリティ確保を優先する
中位	資産が被害にあった場合、組織または個人の活動に重大な影響をあたえるもの 例）資産が被害にあった場合、当該組織の利益が悪化し、事業に影響を及ぼすレベル	• Android OS のセキュリティモデルを活用し、その範囲内で保護する • UX 等の他要素よりもセキュリティ確保を優先する
低位	資産が被害にあった場合、組織または個人の活動に限定的な影響をあたえるもの 例）資産が被害にあった場合、当該組織の利益に影響を与え得るが、他の要素により利益の補てんが可能なレベル	• Android OS のセキュリティモデルを活用し、その範囲内で保護する • UX 等の他要素とセキュリティを比較し、他要素を優先しても良い

本ガイドの対象範囲

本ガイドにおける資産分類と保護施策は、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施策を基準にしている。具体的には、資産分類で中位レベル以下の資産に対して Android OS のセキュリティモデルが機能していることを前提に、Android OS のセキ

 All rights reserved © Japan Smartphone Security Association.

ュリティモデルを活用した保護施策を想定している。一方、高位レベルの資産は、root 権限が奪取された状態からの攻撃や、APK 解析・改造による攻撃といった、Android OS のセキュリティモデルが破られた状態での攻撃からも、保護が必要な資産であると想定している。こ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する。これはガイド文書に簡潔に書けるようなノウハウではないし、個別の状況に応じて適切な防御設計は異なるため本ガイドの対象外としている。root 権限奪取からの攻撃や APK 解析・改造などの高度な攻撃に対する保護が必要な場合は、Android の耐タンパ設計に詳しいセキュリティ専門家に相談することをお勧めする。

3.1.4. センシティブな情報

本ガイドのこれ以降の文章において情報資産を「センシティブな情報」と表現している。前節で述べたようにアプリが扱う個々の情報資産ごとに資産のレベルや保護施策のレベルを判断しなければならない。

All rights reserved © Japan Smartphone Security Association.

3.2. 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、アプリの誤動作を誘発し、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 Internet 経由の InputStream を取得することが保証される。

 All rights reserved © Japan Smartphone Security Association.

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

All rights reserved © Japan Smartphone Security Association.

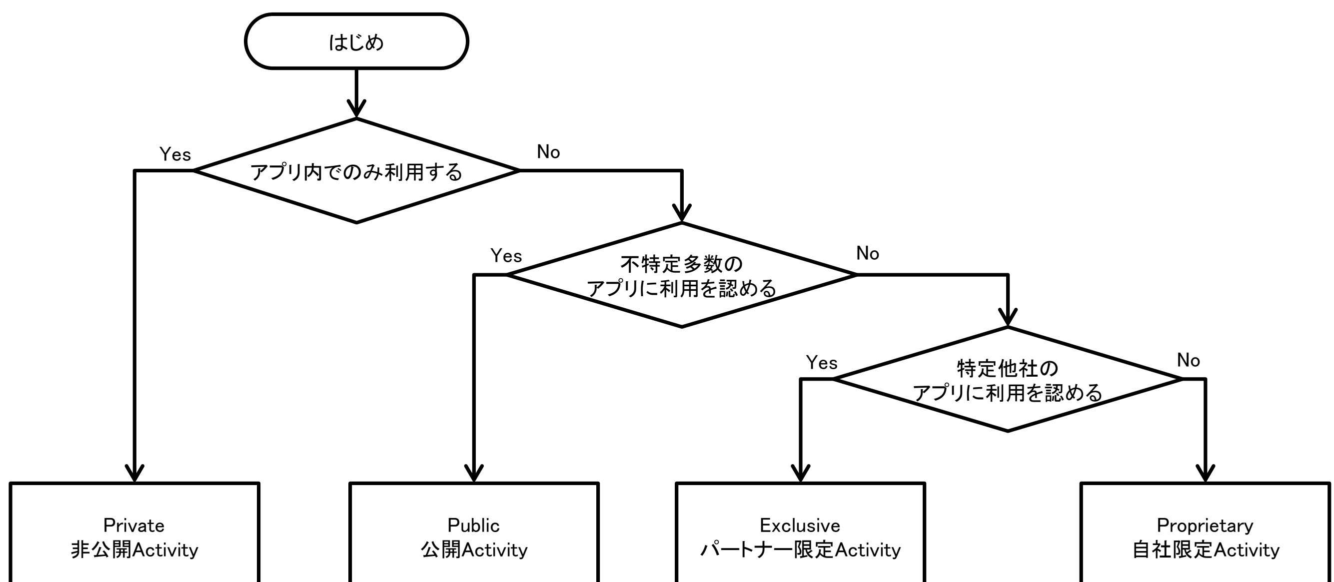
4. 安全にテクノロジーを活用する

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・利用する

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。ここでは、Activity がどのように利用されるかという観点で、Activity を 4 つのタイプに分類した。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。なお、どのような相手を利用するかによって適切な防御手段が決まるため、Activity の利用側の実装についても合わせて説明する。



All rights reserved © Japan Smartphone Security Association.

4.1.1.1. 非公開 Activity を作る・利用する

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。

同一アプリ内だけで利用される Activity(非公開 Activity)を利用する際は、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。ただし、Activity を呼び出す際に使用する Intent は第三者によって読み取られる恐れがある。そのため、Activity に送信する Intent にセンシティブな情報を格納する場合には、その情報が悪意のある第三者に読み取られることのないように、適切な対応を実施する必要がある。

以下に非公開 Activity を作る側のサンプルコードを示す。

ポイント(Activity を作る):

1. taskAffinity を用いてアフィニティを指定しない
2. Activity には launchMode を指定せず、値をデフォルトのまま"standard"とする
3. exported="false"により、明示的に非公開設定する
4. 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
5. 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 exported 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" />

    <!-- ★ポイント 1★ taskAffinity を用いてアフィニティを指定しない -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 1★ taskAffinity を用いてアフィニティを指定しない -->
        <!-- ★ポイント 2★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
        <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★ taskAffinity を用いてアフィニティを指定しない -->
        <!-- ★ポイント 2★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
        <!-- ★ポイント 3★ exported="false"により、明示的に非公開設定する -->
        <activity
```

All rights reserved © Japan Smartphone Security Association.

```
            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 4★ 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。「3.2 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 5★ 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();
    }
}
```

次に非公開 Activity を利用する側のサンプルコードを示す。

ポイント(Activity を利用する)：

6. Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない
7. 同一アプリ内 Activity はクラス指定の明示的 Intent で呼び出す
8. 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[1]
9. 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

[1] ただし、ポイント 1, 2, 6 を遵守している場合を除いては Intent が第三者に読み取られるおそれがあることに注意する必要がある。詳細はルールブックセクションの 4.1.2.2、4.1.2.3、4.1.2.4 を参照すること。

 All rights reserved © Japan Smartphone Security Association.

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 6★ Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない

        // ★ポイント 7★ 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 8★ 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 9★ 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。「3.2 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;
        }
    }
}
```

All rights reserved © Japan Smartphone Security Association.

4.1.1.2. 公開 Activity を作る・利用する

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である。
また、公開 Activity を利用する場合には、送信する Intent がマルウェアに受信される、あるいは読み取られることがあることに注意が必要である。

以下に公開 Activity を作る側のサンプルコードを示す。

ポイント(Activity を作る):
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。「3.2 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();
    }
}
```

次に公開 Activity を利用する側のサンプルコードを示す。

ポイント(Activity を利用する):

3. センシティブな情報を送信してはならない
4. 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 3★ 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");
```

All rights reserved © Japan Smartphone Security Association.

```
            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 4★ 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。「3.2 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;
        }
    }
}
```


All rights reserved © Japan Smartphone Security Association.

4.1.1.3. パートナー限定 Activity を作る・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。

なお、Activity を呼び出す際に使用する Intent は第三者によって読み取られる恐れがある。そのため、Activity に送信する Intent にセンシティブな情報を格納する場合には、その情報が悪意のある第三者に読み取られることのないように、適切な対応を実施する必要がある。

以下にパートナー限定 Activity を作る側のサンプルコードを示す。

ポイント(Activity を作る):
1. taskAffinity を用いてアフィニティを指定しない
2. Activity には launchMode を指定せず、値をデフォルトのまま"standard"とする
3. Intent Filter は定義してはならない
4. 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
5. 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
6. 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、ホワイトリストを用いたアプリの確認方法については、「4.1.3.2 利用元アプリを確認する」を参照すること。また、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" />

    <!-- ★ポイント 1★ taskAffinity を用いてアフィニティを指定しない -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 1★ taskAffinity を用いてアフィニティを指定しない -->
        <!-- ★ポイント 2★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
        <!-- ★ポイント 3★ Intent Filter を定義してはならない -->
        <activity
            android:name=".ExclusiveActivity"
            android:exported="true" />

    </application>
</manifest>
```

All rights reserved © Japan Smartphone Security Association.

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 4★ 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 4★ 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 5★ 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。「3.2 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 6★ 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();
    }
```

 All rights reserved © Japan Smartphone Security Association.

```
}
```

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

All rights reserved © Japan Smartphone Security Association.

```
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

次にパートナー限定 Activity を利用する側のサンプルコードを示す。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(Activity を利用する):

7. taskAffinity を用いてアフィニティを指定しない
8. Activity には launchMode を指定せず、値をデフォルトのまま”standard”とする
9. 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
10. Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない
11. 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
12. 明示的 Intent によりパートナー限定 Activity を呼び出す
13. startActivityForResult()によりパートナー限定 Activity を呼び出す
14. 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、ホワイトリストを用いたアプリの確認方法については、「4.1.3.2 利用元アプリを確認する」を参照すること。また、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"
```

 All rights reserved © Japan Smartphone Security Association.

```
    package="org.jssec.android.activity.exclusiveuser"
    android:versionCode="1"
    android:versionName="1.0" >

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

    <!-- ★ポイント 7★ taskAffinity を用いてアフィニティを指定しない -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 7★ taskAffinity を用いてアフィニティを指定しない -->
        <!-- ★ポイント 8★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
        <activity
            android:name=".ExclusiveUserActivity"
            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>
```

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 9★ 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);
```

All rights reserved © Japan Smartphone Security Association.

```
    }

    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 9★ 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 10★ Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない

            // ★ポイント 11★ 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 12★ 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 13★ 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 14★ 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。「3.2 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;
        }
```


All rights reserved © Japan Smartphone Security Association.

```
    }
}
```

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.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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) {
```

All rights reserved © Japan Smartphone Security Association.

```
        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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

 All rights reserved © Japan Smartphone Security Association.

4.1.1.4. 自社限定 Activity を作る・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。

なお、Activity を呼び出す際に使用する Intent は第三者によって読み取られる恐れがある。そのため、Activity に送信する Intent にセンシティブな情報を格納する場合には、その情報が悪意のある第三者に読み取られることのないように、適切な対応を実施する必要がある。

以下に自社限定 Activity を作る側のサンプルコードを示す。

ポイント(Activity を作る):

1. 独自定義 Signature Permission を定義する
2. taskAffinity を用いてアフィニティを指定しない
3. Activity には launchMode を指定せず、値をデフォルトのまま"standard"とする
4. Activity 定義にて、独自定義 Signature Permission を要求宣言する
5. Intent Filter は定義してはならない
6. 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
7. 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
8. 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
9. 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.activity.proprietaryactivity.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- ★ポイント 2★ taskAffinity を用いてアフィニティを指定しない -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 2★ taskAffinity を用いてアフィニティを指定しない -->
        <!-- ★ポイント 3★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
        <!-- ★ポイント 4★ 独自定義 Signature Permission を要求宣言する -->
        <!-- ★ポイント 5★ Intent Filter を定義してはならない -->
        <activity
            android:name=".ProprietaryActivity"
            android:exported="true"
            android:permission="org.jssec.android.activity.proprietaryactivity.MY_PERMISSION" />
```

All rights reserved © Japan Smartphone Security Association.

```
    </application>
</manifest>
```

PropriearyActivity.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.activity.proprietaryactivity.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 6★ 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 7★ 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。「3.2 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 8★ 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
```

 All rights reserved © Japan Smartphone Security Association.

```
        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;
        }
    }
}
```

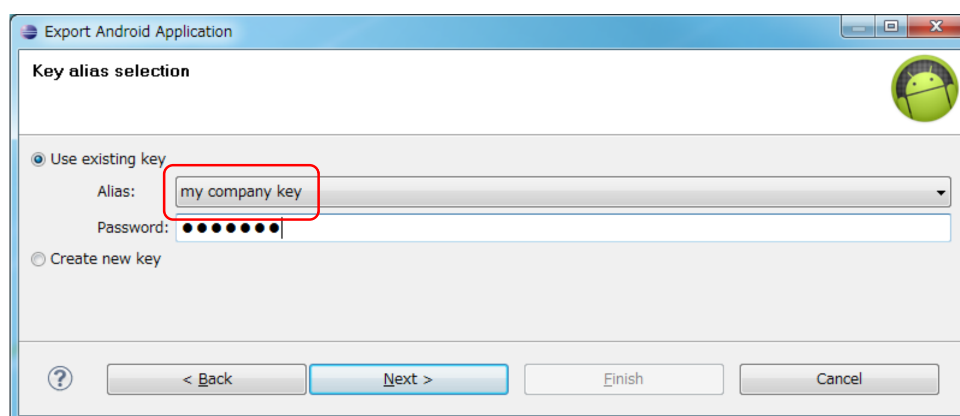
PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;



public class PkgCert {
```

All rights reserved © Japan Smartphone Security Association.

```
    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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

★ポイント 9★ 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。



次に自社限定 Activity を利用する側のサンプルコードを示す。ここでは自社限定 Activity を呼び出す方法を説明する。

All rights reserved © Japan Smartphone Security Association.

ポイント(Activity を利用する):

10. 独自定義 Signature Permission を利用宣言する
11. taskAffinity を用いてアフィニティを指定しない
12. Activity には launchMode を指定せず、値をデフォルトのまま"standard"とする
13. 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
14. 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
15. 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
16. 明示的 Intent により自社限定 Activity を呼び出す
17. 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
18. 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 10★ 独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.activity.proprietaryactivity.MY_PERMISSION" />

    <!-- ★ポイント 11★ taskAffinity を用いてアフィニティを指定しない -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 11★ taskAffinity を用いてアフィニティを指定しない -->
        <!-- ★ポイント 12★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
        <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;
```

All rights reserved © Japan Smartphone Security Association.

```
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.activity.proprietaryactivity.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 13★ 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 14★ 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 15★ 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");
```

 All rights reserved © Japan Smartphone Security Association.

```
            // ★ポイント 16★ 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 17★ 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。「3.2 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);
```

All rights reserved © Japan Smartphone Security Association.

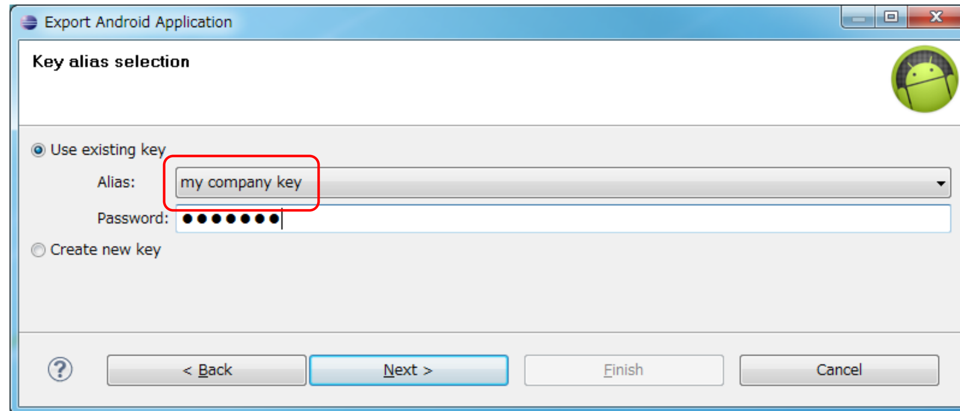
```
        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


All rights reserved © Japan Smartphone Security Association.

★ポイント 18★ 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。



All rights reserved © Japan Smartphone Security Association.

4.1.2. ルールブック

Activity を作る際、または Activity に Intent を送信する際には以下のルールを守ること。

1.	アプリ内でのみ使用する Activity は非公開設定する	（必須）
2.	Activity に対して、taskAffinity を用いてアフィニティを指定しない	（必須）
3.	Activity の起動モードには"standard"を使用する	（必須）
4.	Activity に送信する Intent には FLAG_ACTIVITY_NEW_TASK を設定しない	（必須）
5.	受信 Intent の安全性を確認する	（必須）
6.	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）
7.	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）
8.	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）
9.	利用先 Activity からの戻り Intent の安全性を確認する	（必須）
10.	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）
11.	資産を二次的に提供する場合には、その資産の従来の保護水準を維持する	（必須）
12.	センシティブな情報はできる限り送らない	（推奨）

4.1.2.1. アプリ内でのみ使用する 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。またこのような Activity では開発者も Activity を攻撃する Intent を想定しないことが多い。このような Activity は明示的に非公開設定し、非公開 Activity とする。

AndroidManifest.xml

```
        <!-- 非公開 Activity -->
        <!-- ★ポイント 3★ 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" />
```

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される Activity であり、かつ Intent Filter を設置するような設計はしてはならない。Intent Filter の性質上、同一アプリ内の非公開 Activity を呼び出すつもりでも、Intent Filter 経由で呼び出したときに意図せず他アプリの Activity を呼び出してしまう可能性もある。
詳細は、アドバンスト「4.1.3.1 exported 設定と intent-filter 設定の組み合わせ(Activity の場合)」を参照すること。

AndroidManifest.xml(非推奨)

```
        <!-- 非公開 Activity -->
        <!-- ★ポイント 3★ 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>
```

 All rights reserved © Japan Smartphone Security Association.

```
            <action android:name="org.jssec.android.activity.OPEN />
        </intent-filter>
    </activity>
```

4.1.2.2. Activity の起動モードには"standard"を使用する （必須）

Activity の起動モードとは、Activity を呼び出す際に、Activity のインスタンスの新規生成や、タスクの新規生成を制御するための設定である。デフォルト設定は"standard"である。"standard"設定では、Intent を使って Activity を呼び出すときには常に新規インスタンスを生成し、タスクは呼び出し側 Activity が属するタスクに従い、新規にタスクが生成されることはない。タスクが新規に生成されると、呼び出しに使った Intent が別のアプリから読み取り可能になる。そのため、Activity の起動モードには"standard"を用いるべきである。

Activity の起動モードは AndroidManifest.xml 内にて android:launchMode で明示的に設定可能であるが、上記の理由により、各 Activity に対して android:launchMode を指定せず、値をデフォルトのまま"standard"とするべきである。

AndroidManifest.xml
```
    <!-- ★ポイント 2★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
    <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 2★ Activity には launchMode を指定せず、値をデフォルトのまま" standard" とする -->
    <activity
        android:name=".PrivateActivity"
        android:label="@string/app_name"
        android:exported="false" />
</application>
```

「4.1.3.3 Activity に送信される Intent の読み取り」、「4.1.3.4 ルート Activity について」を参照すること。

4.1.2.3. Activity に対して、taskAffinity を用いてアフィニティを指定しない （必須）

Android では、Activity はタスクによって管理される。タスクの名前は、ルート Activity の持つアフィニティによって決定される。一方でルート以外の Activity に関しては、所属するタスクがアフィニティだけでは決定されず、Activity の起動モードにも依存する。詳細は「4.1.3.4 ルート Activity について」を参照すること。

デフォルト設定では各 Activity はパッケージ名をアフィニティとして持つ。その結果、タスクはアプリごとに割り当てられるので、同一アプリ内の全ての Activity は同一タスクに所属する。タスクの割り当てを変更するには、AndroidManifest.xml への明示的なアフィニティ記述や、Activity に送信する Intent へのフラグ設定をすればよい。

All rights reserved © Japan Smartphone Security Association.

ただし、タスクの割り当てを変更した場合は、異なるタスクに属する Activity に送信した Intent を別アプリによって読み出せる可能性がある。
送信 Intent の内容を読み取られないようにするには、AndroidManifest.xml 内で android:taskAffinity を指定せず、デフォルト(パッケージ名と同一)のままにすべきである。

以下に非公開 Activity の作成側と利用側における AndroidManifest.xml を示す。

AndroidManifest.xml
```
<!-- ★ポイント 1★ taskAffinity を用いてアフィニティを指定しない -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 1★ taskAffinity を用いてアフィニティを指定しない -->
        <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★ taskAffinity を用いてアフィニティを指定しない -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
```

タスクとアフィニティの詳細な解説は、「Google Android プログラミング入門」[2]、あるいは、Google Developers API Guide “Tasks and Back Stack”[3]の解説および「4.1.3.3 Activity に送信される Intent の読み取り」、「4.1.3.4 ルート Activity について」を参照すること。

4.1.2.4. Activity に送信する Intent には FLAG_ACTIVITY_NEW_TASK を設定しない （必須）

Activity の起動モードは startActivity()あるいは startActivityForResult()の実行時にも変更することが可能であり、タスクが新規に生成される場合がある。そのため、Activity の起動モードを実行時に変更しないようにする必要がある。

Activity の起動モードを変更するには、setFlags()や addFlags()を用いて Intent にフラグを設定し、その Intent を startActivity()または startActivityForResult()の引数とする。タスクを新規に生成するためのフラグは

[2] 江川、藤井、麻野、藤田、山田、山岡、佐野、竹端著「Google Android プログラミング入門」[2](アスキー・メディアワークス、2009 年 7 月)
[3] http://developer.android.com/guide/components/tasks-and-back-stack.html

 All rights reserved © Japan Smartphone Security Association.

FLAG_ACTIVITY_NEW_TASK である。FLAG_ACTIVITY_NEW_TASK が設定されると、呼び出された Activity のタスクがバックグラウンドあるいはフォアグラウンド上に存在しない場合に、新規にタスクが生成される。FLAG_ACTIVITY_MULTIPLE_TASK は FLAG_ACTIVITY_NEW_TASK と同時に設定することもできる。この場合には、タスクが必ず新規生成される。どちらの設定もタスクを生成する可能性があるため、Intent に設定しないようにすべきである。

Intent の送信例

```
Intent intent = new Intent();

// ★ポイント 6★ Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない

intent.setClass(this, PrivateActivity.class);
intent.putExtra("PARAM", "センシティブな情報");

startActivityForResult(intent, REQUEST_CODE);
```

なお、Activity に送信する Intent に FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS フラグを明示的に設定することで、タスクが生成されたとしてもその内容が読み取られないようにできると考えるかもしれない。しかしながら、この方法を用いても送信された Intent の内容を読み取ることが可能である。したがって、FLAG_ACTIVITY_NEW_TASK の使用は避けるべきである。

「4.1.3.1 exported 設定と intent-filter 設定の組み合わせ(Activity の場合)」および「4.1.3.3 Activity に送信される Intent の読み取り」、「4.1.3.4 ルート Activity について」も参照すること。

4.1.2.5. 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、受信 Intent のデータを処理する際には、まず受信 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 を無条件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。

「3.2 入力データの安全性を確認する」を参照すること。

4.1.2.6. 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

All rights reserved © Japan Smartphone Security Association.

携する方法」を参照すること。

4.1.2.7. 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()を用いて結果情報を返送する際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6★ 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();
    }
```

4.1.2.8. 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

 All rights reserved © Japan Smartphone Security Association.

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。

「4.1.3.1 exported 設定と intent-filter 設定の組み合わせ(Activity の場合)」も参照すること。

4.1.2.9. 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、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 を無条件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。

「3.2 入力データの安全性を確認する」を参照すること。

4.1.2.10. 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・利用する」を参照すること。また、技術的な詳細に関しては「4.1.3.2 利用元アプリを確認する」を参照すること。

4.1.2.11. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する （必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲(Redelegation)していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4 Permission の再委譲問題」を参照すること。

4.1.2.12. 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

All rights reserved © Japan Smartphone Security Association.

うリスクがある。「4.1.3.5 Activity 利用時のログ出力について」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、センシティブな情報は 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。

また、ルート Activity にはセンシティブな情報を送ってはならない。ルート Activity とは、タスクが生成された時に最初に呼び出された Activity のことである。例えば、ランチャーから起動された Activity は常にルート Activity である。

ルート Activity に関しての詳細は、「4.1.3.3 Activity に送信される Intent の読み取り」、「4.1.3.4 ルート Activity について」も参照すること。

 All rights reserved © Japan Smartphone Security Association.

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(Activity の場合)

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	(使用禁止)	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

「intent-filter 定義がある」&「exported=false」を使用禁止にしているのは、Android の振る舞いに抜け穴があり、Intent Filter の性質上、意図せず他アプリの Activity を呼び出してしまう場合が存在するためである。

以下の 2 つの図は、その説明のためのものである。
図 4.1-1 は、同一アプリ内からしか非公開 Activity(アプリ A)を暗黙的 Intent で呼び出せない正常な動作の例である。Intent-filter(図中 action="X")を定義しているのが、アプリ A しかいないので意図通りの動きとなっている。

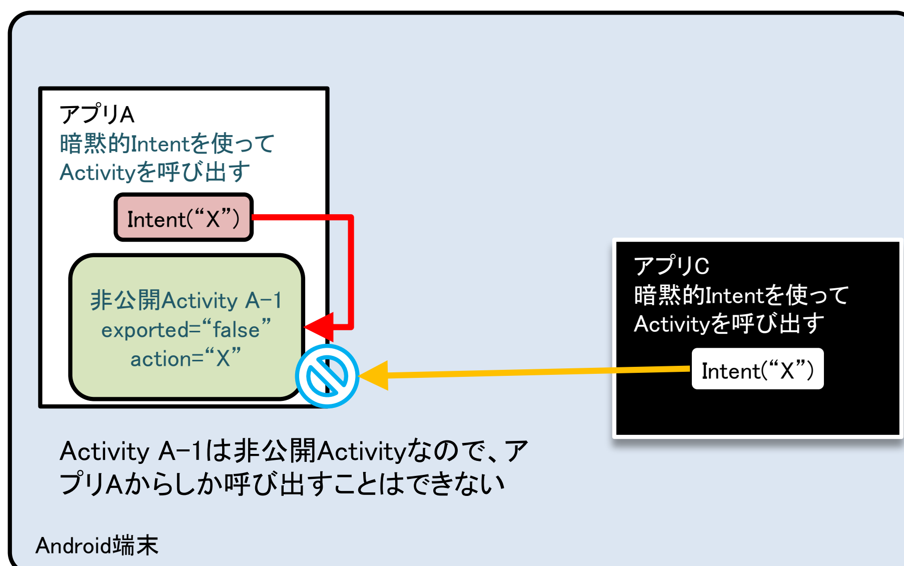


図 4.1-1

図 4.1-2 は、アプリ A に加えてアプリ B でも同じ intent-filter(図中 action="X")を定義している場合である。

図 4.1-2 では、アプリ A が暗黙的 Intent を送信して同一アプリ内の非公開 Activity を呼び出そうとするが、「アプリケーションの選択」ダイアログが表示され、ユーザーの選択によって公開 Activity(B-1)が呼び出されてしまう例を示している。これにより他アプリに対してセンシティブな情報を送信したり、意図せぬ戻り値を受け取る可能性が生じてしまう。

All rights reserved © Japan Smartphone Security Association.

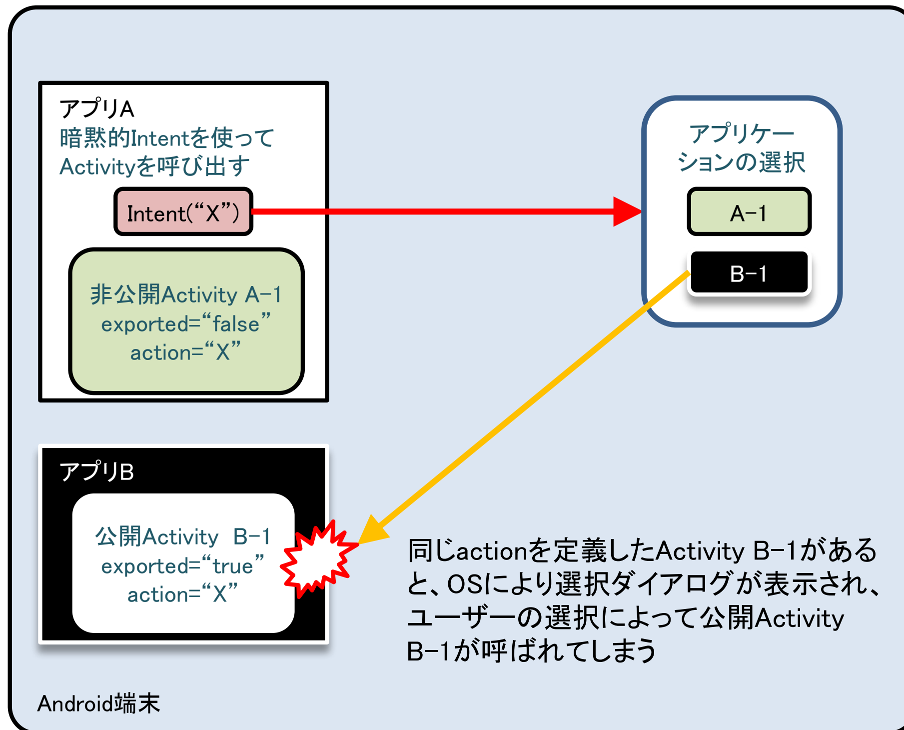


図 4.1-2

このように、Intent Filter を用いた非公開 Activity の暗黙的 Intent 呼び出しは、意図せぬアプリとの情報のやり取りを許してしまうので行うべきではない。
なお、この挙動はアプリ A、アプリ B のインストール順序には依存しないことを確認している。

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。

 All rights reserved © Japan Smartphone Security Association.

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 4★ 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 4★ 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())) {
```

All rights reserved © Japan Smartphone Security Association.

```
            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 5★ 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。「3.2 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 6★ 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;
```

 All rights reserved © Japan Smartphone Security Association.

```
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

4.1.3.3. Activity に送信される Intent の読み取り

タスクのルート Activity に送信された Intent は、タスク履歴に追加される。ルート Activity とはタスクの起点となる Activity のことである。タスク履歴に追加された Intent は、ActivityManager クラスを使うことでどのアプリからも自由に読み出すことが可能である。

All rights reserved © Japan Smartphone Security Association.

アプリからタスク履歴を参照するためのサンプルコードを以下に示す。

タスク履歴を参照するためには、AndroidManifest.xml に GET_TASKS Permission の利用を指定する。

AndroidManifest.xml
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.intent.maliciousactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MaliciousActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <!-- GET_TASKS Permission を指定する -->
    <uses-permission android:name="android.permission.GET_TASKS" />
</manifest>
```

MaliciousActivity.java
```
package org.jssec.android.intent.maliciousactivity;

import java.util.List;

import android.app.Activity;
import android.app.ActivityManager;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

public class MaliciousActivity extends Activity {

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

        // ActivityManager を取得する
        ActivityManager activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
        // タスクの履歴を最新 100 件取得する
        List<ActivityManager.RecentTaskInfo> list = activityManager
                .getRecentTasks(100, ActivityManager.RECENT_WITH_EXCLUDED);
```


All rights reserved © Japan Smartphone Security Association.

```
        for (ActivityManager.RecentTaskInfo r : list) {
            // ルート Activity に送信された Intent を取得し、Log に表示する
            Intent intent = r.baseIntent;
            Log.v("baseIntent", intent.toString());
        }
    }
}
```

ActivityManager クラスの getRecentTasks()により、指定した件数のタスク履歴を取得することができる。各タスクの情報は ActivityManager.RecentTaskInfo クラスのインスタンスに格納されるが、そのメンバー変数 baseIntent には、タスクのルート Activity に送信された Intent が格納されている。ルート Activity とはタスクが生成された時に呼び出された Activity であるので、Activity を呼び出す際には、以下の条件をどちらも満たさないように注意しなければならない。

- Activity が呼び出された際に、タスクが新規に生成される
- 呼び出された Activity がバックグラウンドあるいはフォアグラウンド上に既に存在するタスクのルート Activity である

4.1.3.4. ルート Activity について

ルート Activity とはタスクの起点となる Activity のことである。タスクが生成された時に起動された Activity のことである、と言ってもよい。例えば、デフォルト設定の Activity がランチャーから起動された場合、その Activity はルート Activity となる。Android の仕様によると、ルート Activity に送信される Intent の内容は任意のアプリによって読み取られる恐れがある。そこで、ルート Activity へセンシティブな情報を送信しないように対策を講じる必要がある。本ガイドでは、呼び出された Activity がルートとなるのを防ぐために以下の 3 点をルールに掲げた。

- Activity の起動モードには"standard"を使用する
- taskAffinity を用いてアフィニティを指定しない
- Activity に送信する Intent には FLAG_ACTIVITY_NEW_TASK を設定しない

以下、Activity がルートとなるのはどのような場合かを中心にルート Activity について考察する。

呼び出された Activity がルートとなるための条件に関連するのは、以下に挙げる項目である。

- 呼び出される Activity の起動モード
- 呼び出される Activity のタスクとその起動状態

まず、「呼び出される Activity の起動モード」について説明する。
Activity の起動モードは、AndroidManifest.xml に android:launchMode を記述することで設定できる。記述しない場合は"standard"とみなされる。また、起動モードは Intent に設定するフラグによっても変更可能である。FLAG_ACTIVITY_NEW_TASK フラグは、Activity を"singleTask"モードで起動させる。

指定可能な起動モードは次のとおりである。特に、ルート Activity との関連に焦点を当てて説明する。

- "standard": Activity は呼び出し側タスクとして起動される。したがって、このモードで呼び出された Activity はルートとなることはない。なお、呼び出しの度に Activity のインスタンスが生成される。

All rights reserved © Japan Smartphone Security Association.

- "singleTop": "standard"と同様であるが、フォアグラウンドタスクの最前面に表示されている Activity を起動する場合には、インスタンスが生成されないという点が異なる。
- "singleTask": Activity はアフィニティの値に従って所属するタスクが決まる。Activity のアフィニティと一致するタスクがバックグラウンドあるいはフォアグラウンドに存在しない場合には、タスクが Activity のインスタンスとともに新規に生成される。存在する場合にはどちらも生成されない。前者では、起動された Activity のインスタンスはルートとなる。
- "singleInstance": "singleTask"と同様であるが、次の点で異なる。新規に生成されたタスクには、ルート Activity のみが所属できる点である。したがって、このモードで起動された Activity のインスタンスは常にルートである。ここで注意が必要なのは、呼び出される Activity が持つアフィニティと同じ名前のタスクが既に存在している場合であっても、呼び出される Activity とタスクに含まれる Activity のクラス名が異なる場合である。その場合は、新規にタスクが生成される。

以上より、"singleTask"または"singleInstance"で起動された Activity はルートになる可能性があることが分かる。アプリの安全性を確保するためには、これらのモードで起動しないようにしなければならない。

次に、「呼び出される Activity のタスクとその起動状態」について説明する。たとえ、Activity が"standard"モードで呼び出されたとしても、その Activity が所属するタスクの状態によってルート Activity となる場合がある。

例として、呼び出される Activity のタスクが既にバックグラウンドで起動している場合を考える。問題となるのは、そのタスクの Activity インスタンスが"singleInstance"で起動している場合である。"standard"で呼び出された Activity のアフィニティがタスクと同じだった時に、既存の"singleInstance"の Activity の制限により、新規タスクが生成される。ただし、それぞれの Activity のクラス名が同じ場合は、タスクは生成されず、インスタンスは既存のものが利用される。いずれにしろ、呼び出された Activity はルート Activity になる。

以上のように、ルート Activity が呼び出される条件は実行時の状態に依存するなど複雑である。アプリ開発の際には、"standard"モードで Activity を呼び出すように工夫すべきである。

非公開 Activity に送信される Intent が他アプリから読み取られる例として、非公開 Activity の呼び出し側 Activity を"singleInstance"モードで起動する場合のサンプルコードを以下に示す。このサンプルコードでは、非公開 Activity が"standard"モードで起動されるが、呼び出し側 Activity の"singleInstance"モードの条件により、非公開 Activity は新規タスクのルート Activity となってしまう。この時、非公開 Activity に送信されるセンシティブな情報はタスク履歴に記録されるため、任意のアプリから読み取り可能である。なお、呼び出し側 Activity、非公開 Activity ともに同一のアフィニティを持つ。

AndroidManifest.xml(非推奨)

```
<?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" />
```

 All rights reserved © Japan Smartphone Security Association.

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

        <!-- ルート Activity の起動モードを" singleInstance" とする -->
        <!-- アフィニティは設定せず、アプリのパッケージ名とする -->
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name"
            android:launchMode="singleInstance" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- 非公開 Activity -->
        <!-- 起動モードを" standard" とする -->
        <!-- アフィニティは設定せず、アプリのパッケージ名とする -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
</manifest>
```

非公開 Activity は、受信した Intent に対して結果を返すのみである。

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);

        // 受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 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) {
        Intent intent = new Intent();
        intent.putExtra("RESULT", "センシティブな情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

All rights reserved © Japan Smartphone Security Association.

非公開 Activity の呼び出し側では、Intent にフラグを設定せずに、"standard"モードで非公開 Activity を起動している。

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) {

        // 非公開 Activity を"standard"モードで起動する
        Intent intent = new Intent();
        intent.setClass(this, PrivateActivity.class);
        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");

            // 受信データの安全性を確認する
            // サンプルにつき割愛。「3.2 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;
        }
    }
}
```

4.1.3.5. Activity 利用時のログ出力について

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

 All rights reserved © Japan Smartphone Security Association.

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 Extras 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

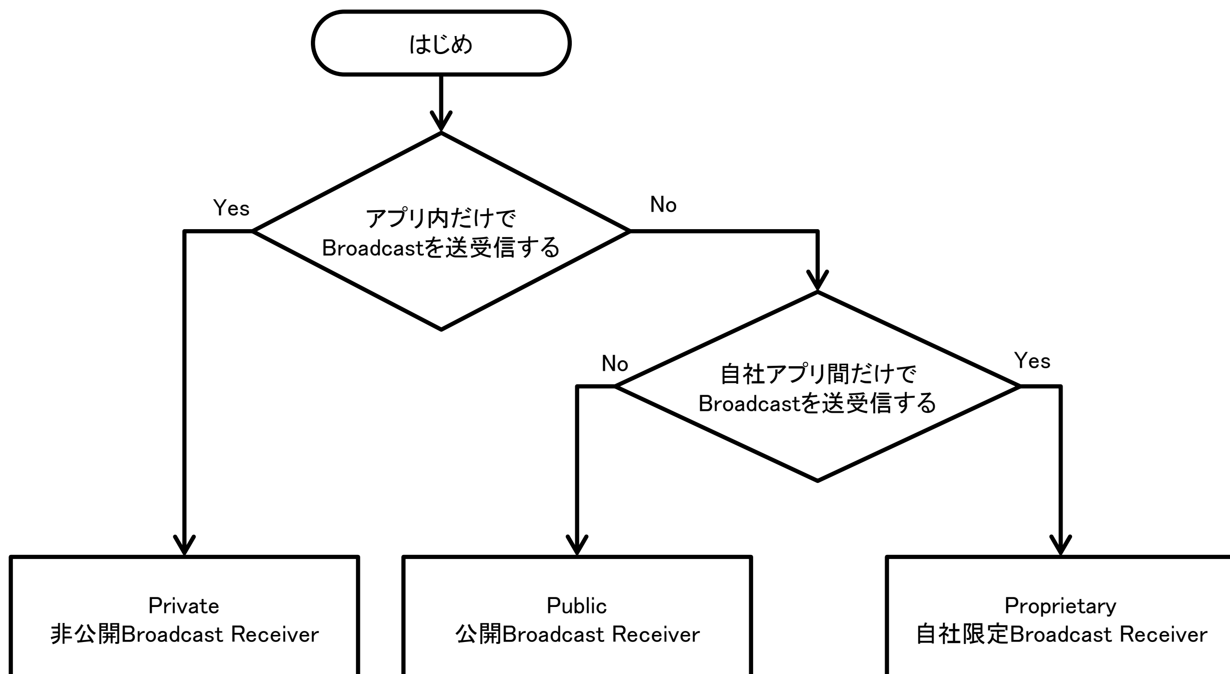
```
        // Extra に設定した内容は LogCat に出力されない
        Uri uri = Uri.parse("mailto:");
        Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
        intent.putExtra(Intent.EXTRA_EMAIL, new String[] {"test@gmail.com"});
        startActivity(intent);
```

All rights reserved © Japan Smartphone Security Association.

4.2. Broadcast を受信する・送信する

4.2.1. 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。ちなみに、パートナー限定の連携に必要な Broadcast 送信元アプリのパッケージ名を受信側アプリで確認する手段がないため、パートナー限定 Broadcast Receiver を作る事はできない。



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。
なお、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決まるため、送信側の実装についても合わせて説明する。

	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	● 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

 All rights reserved © Japan Smartphone Security Association.

	Broadcast Receiver を登録／登録解除する	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

4.2.1.1. 非公開 Broadcast Receiver - Broadcast を受信する・送信する

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 Broadcast 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(Broadcast を受信する):

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;
```

All rights reserved © Japan Smartphone Security Association.

```
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。「3.2 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();
    }
}
```

次に非公開 Broadcast Receiver へ Broadcast 送信するサンプルコードを示す。
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(Broadcast を送信する):

4. 同一アプリ内 Receiver はクラス指定の明示的 Intent で Broadcast 送信する
5. 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
6. 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 4★ 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 5★ 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) {
```

 All rights reserved © Japan Smartphone Security Association.

```
        // ★ポイント 4★ 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 5★ 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 6★ 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。「3.2 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");
    }
}
```

All rights reserved © Japan Smartphone Security Association.

4.2.1.2. 公開 Broadcast Receiver – Broadcast を受信する・送信する

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(Broadcast を受信する):

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 Broadcast Receiver のサンプルコードである Public Receiver 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。「3.2 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();
    }
}
```

 All rights reserved © Japan Smartphone Security Association.

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;
```

All rights reserved © Japan Smartphone Security Association.

```
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 Broadcast Receiver より動的 Broadcast Receiver 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);
    }
```

 All rights reserved © Japan Smartphone Security Association.

```
    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);
    }
}
```

次に公開 Broadcast Receiver へ Broadcast 送信するサンプルコードを示す。
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(Broadcast を送信する):

3. センシティブな情報を送信してはならない
4. 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 3★ 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 3★ 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 3★ 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");
```

All rights reserved © Japan Smartphone Security Association.

```
        sendStickyBroadcast(intent);
    }

    public void onSendStickyOrderedClick(View view) {
        // ★ポイント 3★ 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 4★ 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。「3.2 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");
    }
}
```


All rights reserved © Japan Smartphone Security Association.

4.2.1.3. 自社限定 Broadcast Receiver – Broadcast を受信する・送信する

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(Broadcast を受信する):

1. Broadcast 受信用の独自定義 Signature Permission を定義する
2. 結果受信用の独自定義 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

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

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.broadcast.proprietaryreceiver.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 =
```

All rights reserved © Japan Smartphone Security Association.

```
        "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。「3.2 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();
    }
}
```

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.proprietaryreceiver"
    android:versionCode="1"
    android:versionName="1.0" >

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

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

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

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


All rights reserved © Japan Smartphone Security Association.

```
        <!-- ★ポイント 3★ 静的 Broadcast Receiver 定義にて、独自定義 Signature Permission を要求宣言する -->
        <receiver
            android:name=".ProprietaryReceiver"
            android:permission="org.jssec.android.broadcast.proprietaryreceiver.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>
```

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 ProprietaryReceiverActivity 上にボタンを配置している。動的 Broadcast Receiver インスタンスは ProprietaryReceiverActivity より生存期間が長いため ProprietaryReceiverActivity 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 ProprietaryReceiverActivity 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。

ProprietaryReceiverActivity.java

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

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

public class ProprietaryReceiverActivity extends Activity {
    @Override
    public 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) {
```

All rights reserved © Japan Smartphone Security Association.

```
        Intent intent = new Intent(this, DynamicReceiverService.class);
        stopService(intent);
    }}
```

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 Broadcast Receiver より動的 Broadcast Receiver を優先させる

        // ★ポイント 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
        registerReceiver(mReceiver, filter, "org.jssec.android.broadcast.proprietaryreceiver.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();
    }
}
```

SigPerm.java

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


All rights reserved © Japan Smartphone Security Association.

```
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

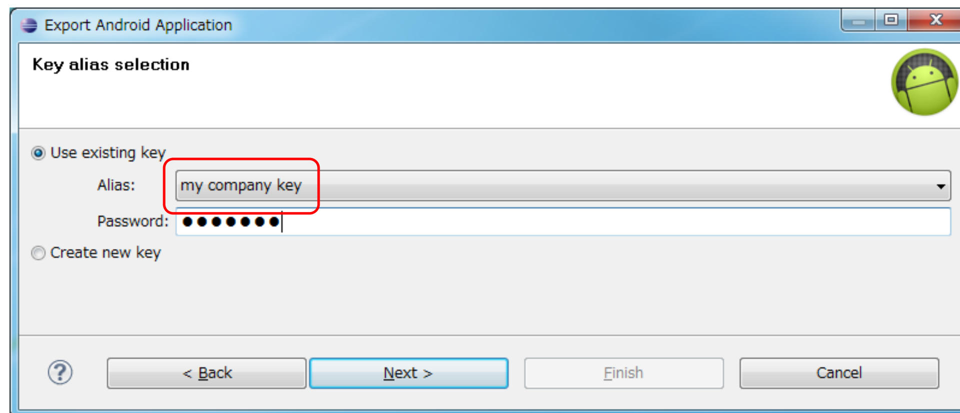
```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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 {
```

All rights reserved © Japan Smartphone Security Association.

```
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

 All rights reserved © Japan Smartphone Security Association.

★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。



次に自社限定 Broadcast Receiver へ Broadcast 送信するサンプルコードを示す。

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(Broadcast を送信する):

9. 結果受信用の独自定義 Signature Permission を定義する
10. Broadcast 受信用の独自定義 Signature Permission を利用宣言する
11. 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
12. 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
13. Receiver に独自定義 Signature Permission を要求する
14. 結果を受け取る場合、結果データの安全性を確認する
15. 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 9★ 結果受信用の独自定義 Signature Permission を定義する -->
    <permission
        android:name="org.jssec.android.broadcast.proprietarysender.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- ★ポイント 10★ Broadcast 受信用の独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.broadcast.proprietaryreceiver.MY_PERMISSION" />

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

All rights reserved © Japan Smartphone Security Association.

```
            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.broadcast.proprietarysender.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 11★ 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 12★ 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
```

 All rights reserved © Japan Smartphone Security Association.

```
        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 13★ Receiver に独自定義 Signature Permission を要求する
        sendBroadcast(intent, "org.jssec.android.broadcast.proprietarysender.MY_PERMISSION");
    }

    public void onSendOrderedClick(View view) {

        // ★ポイント 11★ 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 12★ 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 13★ Receiver に独自定義 Signature Permission を要求する
        sendOrderedBroadcast(intent, "org.jssec.android.broadcast.proprietarysender.MY_PERMISSION",
                mResultReceiver, null, 0, null, null);
    }

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

            // ★ポイント 14★ 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。「3.2 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;
```

All rights reserved © Japan Smartphone Security Association.

```
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;
        }
    }
}
```

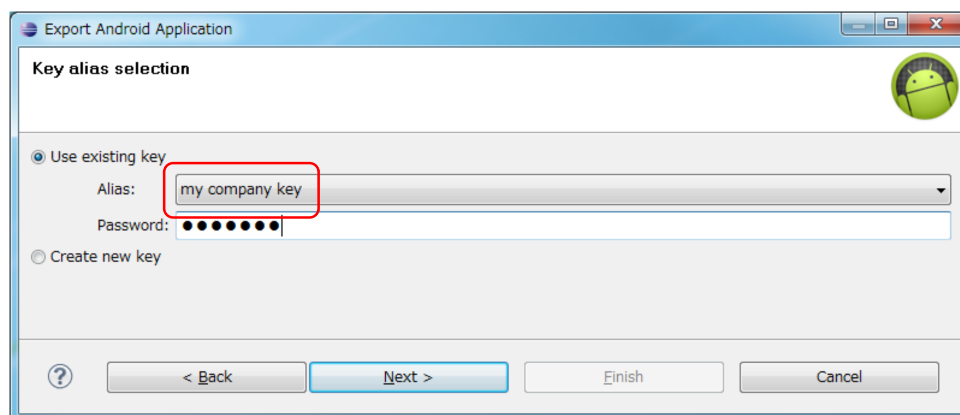
PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
```

 All rights reserved © Japan Smartphone Security Association.

```
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

★ポイント 15★ 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。



All rights reserved © Japan Smartphone Security Association.

4.2.2. ルールブック

Broadcast を送受信する際には以下のルールを守ること。

1.	アプリ内でのみ使用する 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）
5.	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）
6.	Sticky Broadcast にはセンシティブな情報を含めない	（必須）
7.	receiverPermission パラメータの指定なし Ordered Broadcast は届かないことがあることに注意	（必須）
8.	Broadcast Receiver からの返信データの安全性を確認する	（必須）
9.	資産を二次的に提供する場合には、その資産の従来の保護水準を維持する	（必須）

4.2.2.1. アプリ内でのみ使用する 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。

また、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される Receiver であり、かつ Intent Filter を設置するような設計はしてはならない。Intent Filter の性質上、同一アプリ内の非公開 Receiver を呼び出すつもりでも、Intent Filter 経由で呼び出したときに意図せず他アプリの公開 Receiver を呼び出してしまう場合が存在するからである。

AndroidManifest.xml(非推奨)
```
<!-- ポイント 1: 外部アプリに非公開とする Broadcast Receiver -->
<!-- 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>
```

「4.2.3.1 exported 設定と intent-filter 設定の組み合わせ(Receiver の場合)」も参照すること。

4.2.2.2. 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、受信 Intent のデータを処理する際には、まず受信 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

 All rights reserved © Japan Smartphone Security Association.

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 を無条件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。

「3.2 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。
また独自定義 Signature Permission を receiverPermission パラメータに指定して Broadcast 送信する場合も同様に確認する必要がある。

4.2.2.4. 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。

4.2.2.5. 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 Broadcast 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「4.2.1.1 非公開 Broadcast Receiver –

All rights reserved © Japan Smartphone Security Association.

Broadcast」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 パラメータに独自定義 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「4.2.1.3 自社限定 Broadcast Receiver」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「4.2.2.3 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）」も適用しなければならない。

4.2.2.6. 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 パラメータを指定した 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多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。

4.2.2.7. receiverPermission パラメータの指定なし Ordered Broadcast は届かないことがあることに注意（必須）

receiverPermission パラメータ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。

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

結果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、基本的には受信した結果データが攻撃データである可能性を考慮して安全に処理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結果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結果データとして転送することがあるため、結果データを無条件に安全であると考えてはならない。返信

 All rights reserved © Japan Smartphone Security Association.

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、やはり結果データが攻撃データである可能性を考慮して安全に処理しなければならない。

「3.2 入力データの安全性を確認する」を参照すること。

4.2.2.9. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する （必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4 Permission の再委譲問題」を参照すること。

4.2.3. アドバンスト

4.2.3.1. exported 設定と intent-filter 設定の組み合わせ(Receiver の場合)

このガイド文書では、Receiver の用途から非公開 Receiver、公開 Receiver、自社限定 Receiver の 3 タイプの Receiver 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 Receiver 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	(使用禁止)	公開
intent-filter 定義がない	公開、自社限定	非公開	非公開

「intent-filter 定義がある」&「exported=false」を使用禁止にしているのは、Android の振る舞いとして、同一アプリ内の非公開 Receiver に向けて Broadcast を送信したつもりでも、意図せず他アプリの公開 Receiver を呼び出してしまう場合が存在するためである。

以下の 2 つの図で Android の振る舞いによる意図せぬ呼び出しが起こる様子を説明する。
図 4.2-1 は、同一アプリ内からしか非公開 Receiver(アプリ A)を暗黙的 Intent で呼び出せない正常な動作の例である。Intent-filter(図中 action="X")を定義しているのが、アプリ A しかいないので意図通りの動きとなっている。

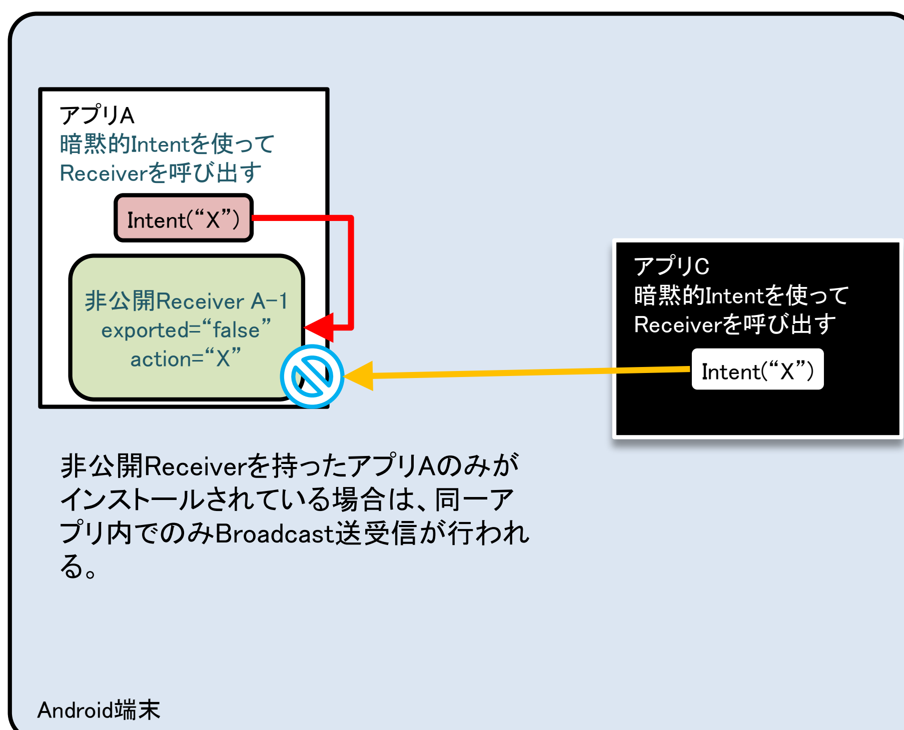


図 4.2-1

図 4.2-2 は、アプリ A に加えてアプリ B でも同じ intent-filter(図中 action="X")を定義している場合である。

 All rights reserved © Japan Smartphone Security Association.

まず、他のアプリ(アプリ C)が暗黙的 Intent で Broadcast を送信するのは、非公開 Receiver(A-1)側は受信をしないので特にセキュリティ的には問題にならない(図の橙色の矢印)。

セキュリティ面で問題になるのは、アプリ A による同一アプリ内の非公開 Receiver の呼び出しである。アプリ A が暗黙的 Intent を Broadcast すると、同一アプリ内の非公開 Receiver に加えて、同じ Intent-filter を定義した B の持つ公開 Receiver(B-1)もその Intent を受信できてしまうからである(図の赤色の矢印)。A からアプリ B に対してセンシティブな情報を送信する可能性が生じてしまう。アプリ B がマルウェアであれば、そのままセンシティブな情報の漏洩に繋がる。

また、Broadcast が Ordered Broadcast であった場合は、意図しない結果情報を受け取ってしまう可能性もある。

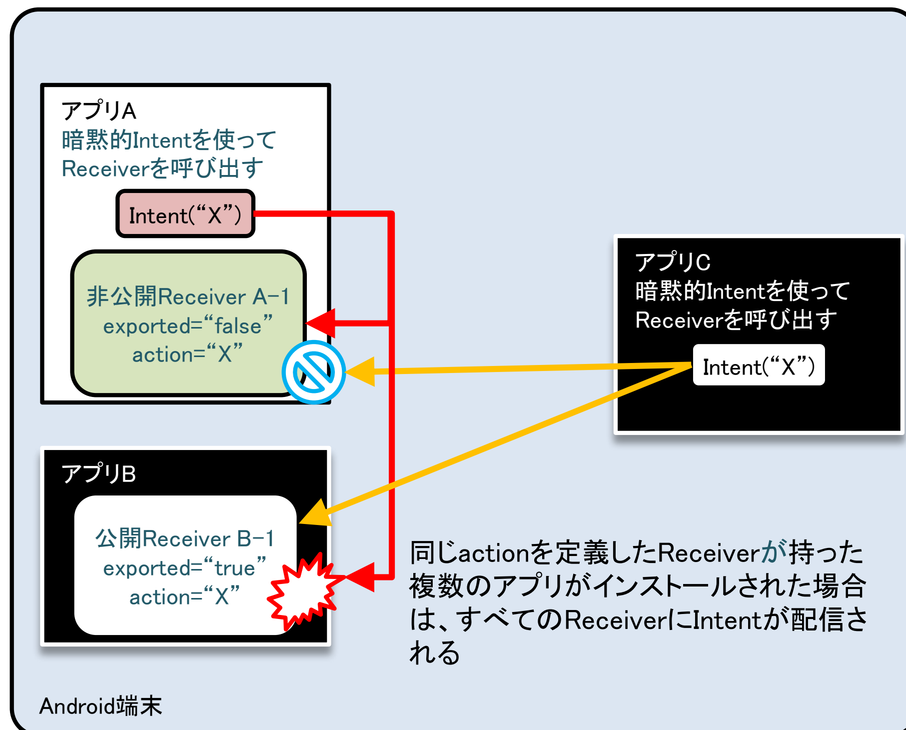


図 4.2-2

4.2.3.2. 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。

4.2.3.3. 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

All rights reserved © Japan Smartphone Security Association.

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。

4.2.3.4. 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 Broadcast	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

 All rights reserved © Japan Smartphone Security Association.

	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。Sticky Broadcast と同様、特定の Permission を持つアプリの 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	–	–	○	○

4.2.3.5. 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

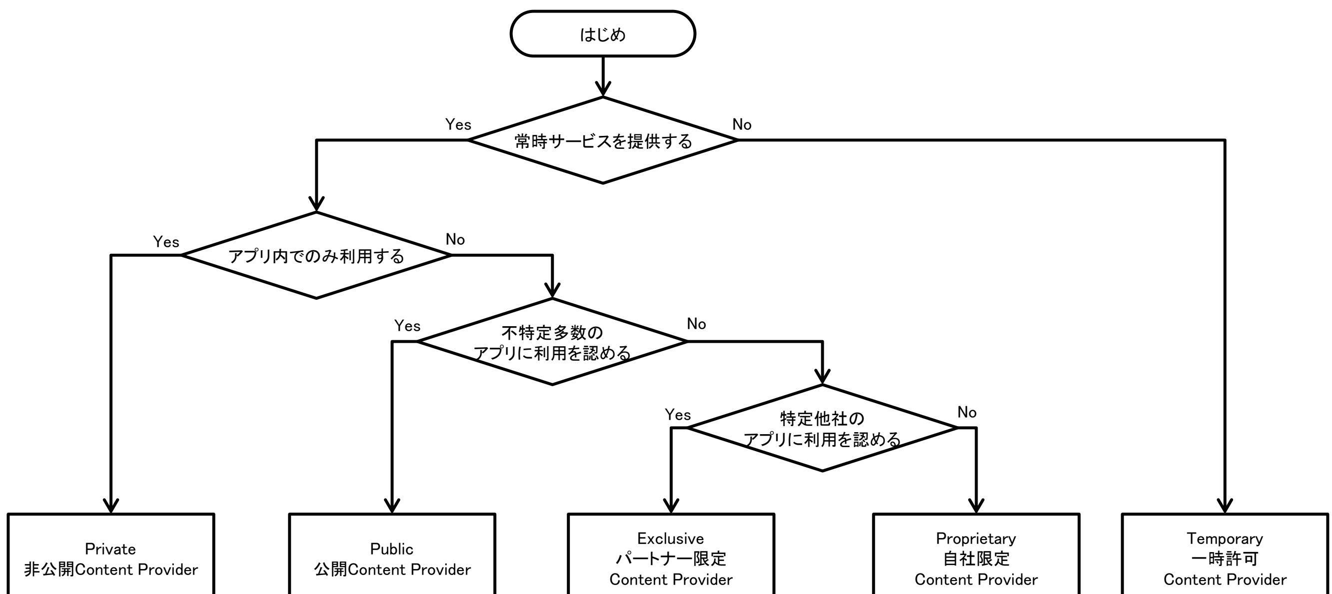
```
W/ActivityManager(275): Permission Denial: receiving Intent { act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION } to org.jssec.android.broadcastreceiver.creating requires org.jssec.android.permission.MY_PERMISSION due to sender org.jssec.android.broadcast.sending (uid 10158)
```

All rights reserved © Japan Smartphone Security Association.

ContentResolver と SQLiteDatabase のインター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ーフェースを規定するだけで、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。なお、ここで紹介するサンプルコードにはデータを保存する処理を含まないので、必要に応じて追加すること。

4.3.1. 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。



 All rights reserved © Japan Smartphone Security Association.

4.3.1.1. 非公開 Content Provider を作る・利用する

非公開 Content Provider は、同一アプリ内だけで利用される Content Provider であり、もっとも安全性の高い Content Provider である。ただし、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。

以下、非公開 Content Provider の実装例を示す。

ポイント(Content Provider を作る)：

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;
```

All rights reserved © Japan Smartphone Security Association.

```
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class PrivateProvider extends ContentProvider {

    public static final String AUTHORITY = "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 が提供するインターフェースを公開
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + 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(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, 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。
        // 「3.2 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。
```

 All rights reserved © Japan Smartphone Security Association.

```
        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。「3.2 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。「3.2 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) {
```

All rights reserved © Japan Smartphone Security Association.

```
        // ★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。「3.2 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;   // update されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        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。「3.2 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;  // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

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

次に、非公開 Content Provider を利用する Activity の例を示す。

 All rights reserved © Japan Smartphone Security Association.

ポイント(Content Provider を利用する):

5. 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
6. 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 5★ 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 = null;
        try {
            cursor = getContentResolver().query(
                    PrivateProvider.Download.CONTENT_URI, null, null, null, null);

            // ★ポイント 6★ 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。「3.2 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();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 5★ 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 6★ 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。「3.2 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]");
```

All rights reserved © Japan Smartphone Security Association.

```
        // ★ポイント 5★ 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 6★ 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。「3.2 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 5★ 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 6★ 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。「3.2 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");
    }
}
```

 All rights reserved © Japan Smartphone Security Association.

4.3.1.2. 公開 Content Provider を作る・利用する

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。クライアントを限定しないことにより、マルウェアから select()によって保持しているデータを抜き取られたり、update()によってデータを書き換えられたり、insert()/delete()によって偽のデータの挿入やデータの削除といった攻撃を受けたりして改ざんされ得ることに注意が必要だ。
また、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。

以下、公開 Content Provider の実装例を示す。

ポイント(Content Provider を作る):
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 AUTHORITY = "org.jssec.android.provider.publicprovider";
```

All rights reserved © Japan Smartphone Security Association.

```
    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 が提供するインターフェースを公開
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + 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(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, 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);
        }
```

 All rights reserved © Japan Smartphone Security Association.

```
    }

    @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。「3.2 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。「3.2 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。「3.2 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。
```

All rights reserved © Japan Smartphone Security Association.

```
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // update されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        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。「3.2 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;  // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

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

次に、公開 Content Provider を利用する Activity の例を示す。

ポイント(Content Provider を利用する)：

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

PublicUserActivity.java

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

 All rights reserved © Japan Smartphone Security Association.

```
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 AUTHORITY = "org.jssec.android.provider.publicprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

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

        // ★ポイント 3★ 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 = null;
        try {
            cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 4★ 結果データの安全性を確認する
            // サンプルにつき割愛。「3.2 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();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

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

All rights reserved © Japan Smartphone Security Association.

```
        // ★ポイント 3★ 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 4★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 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 3★ 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 4★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 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 3★ 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 4★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 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);
    }
```

 All rights reserved © Japan Smartphone Security Association.

```
    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");
    }
}
```

All rights reserved © Japan Smartphone Security Association.

4.3.1.3. パートナー限定 Content Provider を作る・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。

以下、パートナー限定 Content Provider の実装例を示す。

ポイント(Content Provider を作る)：
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;
```

 All rights reserved © Japan Smartphone Security Association.

```
public class ExclusiveProvider extends ContentProvider {

    public static final String AUTHORITY = "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 が提供するインターフェースを公開
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + 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(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, 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。
        // 「3.2 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;
```

 All rights reserved © Japan Smartphone Security Association.

```
        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。
        // 「3.2 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。
        // 「3.2 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;   // update されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;
```

All rights reserved © Japan Smartphone Security Association.

```
        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。
        // 「3.2 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;  // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

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

次に、パートナー限定 Content Provider を利用する Activity の例を示す。

ポイント(Content Provider を利用する):

4. 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
5. 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
6. 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;
```

 All rights reserved © Japan Smartphone Security Association.

```
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 AUTHORITY = "org.jssec.android.provider.exclusiveprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // ★ポイント 4★ 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 4★ 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 5★ 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 = null;
        try {
            cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);
```

All rights reserved © Japan Smartphone Security Association.

```
            // ★ポイント 6★ 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。「3.2 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();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 4★ 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 5★ 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 6★ 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。「3.2 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 4★ 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 5★ 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 6★ 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。「3.2 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
```

 All rights reserved © Japan Smartphone Security Association.

```
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // ★ポイント 4★ 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 5★ 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 6★ 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。「3.2 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;
```

All rights reserved © Japan Smartphone Security Association.

```
    }

    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.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
```

 All rights reserved © Japan Smartphone Security Association.

```
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

All rights reserved © Japan Smartphone Security Association.

4.3.1.4. 自社限定 Content Provider を作る・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。

以下、自社限定 Content Provider の実装例を示す。

ポイント(Content Provider を作る)：

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.provider.proprietaryprovider.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.provider.proprietaryprovider.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 AUTHORITY = "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 が提供するインターフェースを公開
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + 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(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, 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.provider.proprietaryprovider.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";
```

All rights reserved © Japan Smartphone Security Association.

```
            }
        }
        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。
        // 「3.2 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()))) {
```

 All rights reserved © Japan Smartphone Security Association.

```
            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。
        // 「3.2 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。
        // 「3.2 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;   // update されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        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。");
        }
```

All rights reserved © Japan Smartphone Security Association.

```
        // ★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。
        // 「3.2 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;  // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        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;
        }
```

 All rights reserved © Japan Smartphone Security Association.

```
    }
}
```

PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

All rights reserved © Japan Smartphone Security Association.

次に、自社限定 Content Provider を利用する Activity の例を示す。

ポイント(Content Provider を利用する):

6. 独自定義 Signature Permission を利用宣言する
7. 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
8. 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
9. 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
10. 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 6★ 独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.provider.proprietaryprovider.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;
```

 All rights reserved © Japan Smartphone Security Association.

```
public class ProprietaryUserActivity extends Activity {

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

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.provider.proprietaryprovider.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 8★ 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 9★ 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 10★ 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 = null;
        try {
            cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 11★ 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。「3.2 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();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 8★ 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 9★ 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 10★ 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 11★ 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。「3.2 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 8★ 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 9★ 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;
        }
```

 All rights reserved © Japan Smartphone Security Association.

```
        // ★ポイント 10★ 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 11★ 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。「3.2 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 8★ 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 9★ 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 10★ 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 11★ 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。「3.2 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;
```

All rights reserved © Japan Smartphone Security Association.

```java
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

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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);
```

 All rights reserved © Japan Smartphone Security Association.

```
            if (pkginfo.signatures.length != 1) return null;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

4.3.1.5. 一時許可 Content Provider を作る・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。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。

以下、一時許可 Content Provider の実装例を示す。

All rights reserved © Japan Smartphone Security Association.

ポイント(Content Provider を作る):

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. 一時的に許可した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. 一時的に許可したアプリに開示してよい情報に限り返送してよい
6. 一時的にアクセスを許可する URI を Intent に指定する
7. 一時的に許可するアクセス権限を Intent に指定する
8. 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
9. 一時許可の要求元アプリに Intent を返信する

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>
```

 All rights reserved © Japan Smartphone Security Association.

TemporaryProvider.java

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

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 TemporaryProvider extends ContentProvider {
    public static final String AUTHORITIY = "org.jssec.android.provider.temporaryprovider";
    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 が提供するインターフェースを公開
    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) {
```

All rights reserved © Japan Smartphone Security Association.

```
        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 4★ 一時的に許可した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。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 5★ 一時的に許可した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 4★ 一時的に許可した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。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 5★ 一時的に許可した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,
```

 All rights reserved © Japan Smartphone Security Association.

```
            String[] selectionArgs) {

        // ★ポイント 4★ 一時的に許可した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。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 5★ 一時的に許可した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;   // update されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

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

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

        // ★ポイント 4★ 一時的に許可した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。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 5★ 一時的に許可した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;  // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

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

TemporaryActiveGrantActivity.java

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

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
```

All rights reserved © Japan Smartphone Security Association.

```
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 6★ 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 7★ 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 8★ 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 6★ 一時的にアクセスを許可する URI を Intent に指定する
```


All rights reserved © Japan Smartphone Security Association.

```
        intent.setData(TemporaryProvider.Address.CONTENT_URI);

        // ★ポイント 7★ 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 9★ 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();
    }
}
```

次に、一時許可 Content Provider を利用する Activity の例を示す。

ポイント(Content Provider を利用する):

10. センシティブな情報をリクエストに含めてはならない
11. 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.ActivityNotFoundException;
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;
import android.widget.Toast;

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 AUTHORITY = "org.jssec.android.provider.temporaryprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    private static final int REQUEST_CODE = 1;

    public void onQueryClick(View view) {

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

```
        Cursor cursor = null;
        try {
            if (!providerExists(Address.CONTENT_URI)) {
                logLine("  Content Provider が不在");
                return;
            }

            // ★ポイント 10★ 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 = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 11★ 結果データの安全性を確認する
            // サンプルにつき割愛。「3.2 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();
                }
            }
        } catch (SecurityException ex) {
            logLine("  例外:" + ex.getMessage());
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    // 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);
        try {
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            logLine("Grant の要求に失敗しました。¥nTemporaryProvider がインストールされているか確認してください。"
);
        }
    }

    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);
```

 All rights reserved © Japan Smartphone Security Association.

```
            mLogView.append("¥n");
        }
}
```

All rights reserved © Japan Smartphone Security Association.

4.3.2. ルールブック

Content Provider の実装時には以下のルールを守ること。

1.	Android 2.2(API Level 8)以前ではアプリ内でのみ使用する Content Provider は作らない	(必須)
2.	アプリ内でのみ使用する 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)
6.	資産を二次的に提供する場合には、その資産の従来の保護水準を維持する	(必須)

また、利用側は、以下のルールも守ること。

7.	Content Provider の結果データの安全性を確認する	(必須)

4.3.2.1. Android 2.2(API Level 8)以前ではアプリ内でのみ使用する Content Provider は作らない (必須)

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。

4.3.2.2. アプリ内でのみ使用する 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" />
```

4.3.2.3. 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、リクエストパラメータを処理する際には、まずその安全性を確認しなければならない。

 All rights reserved © Japan Smartphone Security Association.

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ー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、リクエストを無条件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。

「3.2 入力データの安全性を確認する」を参照すること。

4.3.2.4. 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。

4.3.2.5. 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。

4.3.2.6. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する （必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4 Permission の再委譲問題」を参照すること。

4.3.2.7. 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、結果データを処理する際には、まず結果データの安全性を確認しなければならない。

All rights reserved © Japan Smartphone Security Association.

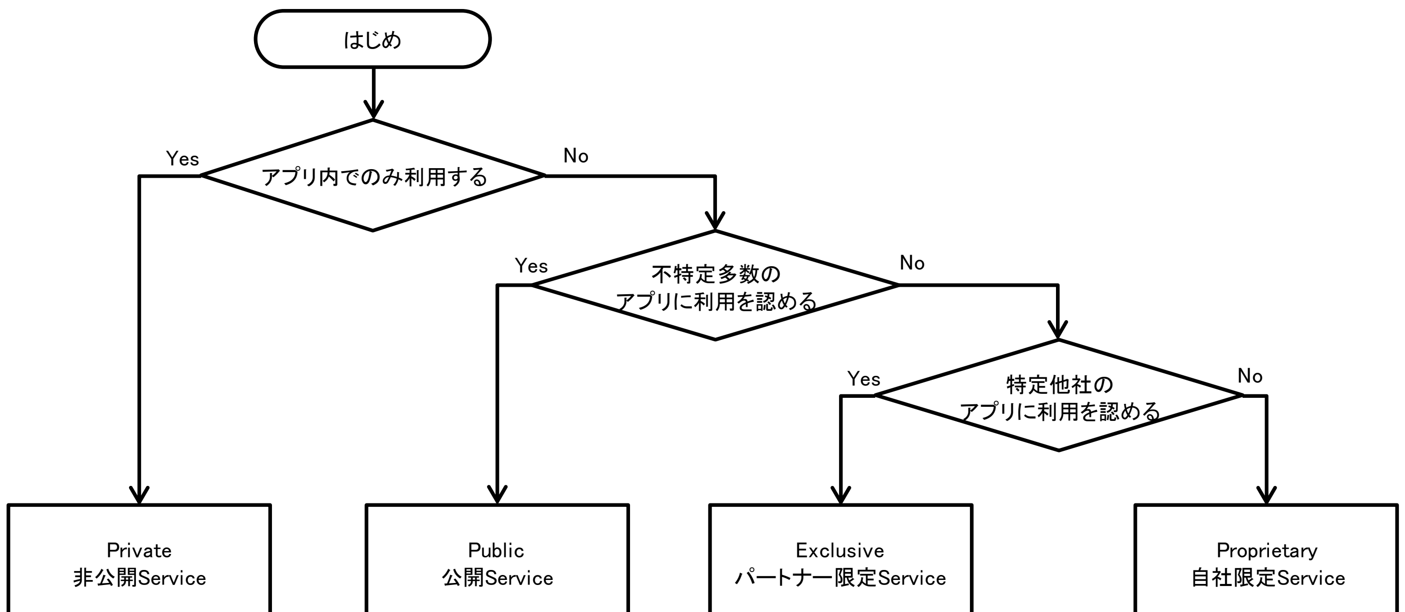
利用先 Content Provider が公開 Content Provider の場合、公開 Content Provider に成り済ましたマルウェアが攻撃結果データを返送してくる可能性がある。利用先 Content Provider が非公開 Content Provider の場合、同一アプリ内から結果データを受け取るのでリスクは少ないが、結果データを無条件に安全であると考えてはならない。その他の Content Provider についても、やはり結果データの安全性を確認する必要がある。

「3.2 入力データの安全性を確認する」を参照すること。

All rights reserved © Japan Smartphone Security Association.

4.4.1. サンプルコード

Service がどのように利用されるかによって、Service 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 Service がどのタイプであるかを判断できる。なお、作成する Service のタイプによって Service を利用する側の実装も決まるので、利用側の実装についても合わせて説明する。



Service には複数の実装方法があり、その中から作成する Service のタイプに合った方法を選択することになる。下表の縦の項目が本文書で扱う実装方法であり、5 種類に分類した。表中の○印は実現可能な組み合わせを示し、その他は実現不可能もしくは困難なものを示す。

なお、Service の実装方法の詳細については、「4.4.3.2 Service の実装方法について」および各 Service タイプのサンプルコード（表中で*印の付いたもの）を参照すること。

表 4.4.1-1

分類	非公開 Service	公開 Service	パートナー限定 Service	自社限定 Service
startService 型	○*	○	–	○
IntentService 型	○	○*	–	○
local bind 型	○	–	–	–
Messenger bind 型	○	○	–	○*
AIDL bind 型	○	○	○*	○

以下では表 4.4.1-1 中の*印の組み合わせを使って各セキュリティタイプの Service のサンプルコードを示す。

All rights reserved © Japan Smartphone Security Association.

4.4.1.1. 非公開 Service を作る・利用する

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

以下、startService 型の Service を使用した例を示す。

ポイント(Service を作る)：

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. 結果を返す場合、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.service.privateservice"
    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 Service -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <service android:name=".PrivateStartService" android:exported="false"/>

        <!-- IntentService を継承した Service -->
        <!-- 非公開 Service -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <service android:name=".PrivateIntentService" android:exported="false"/>

        </application>

</manifest>
```

PrivateStartService.java

```
package org.jssec.android.service.privateservice;

import android.app.Service;
import android.content.Intent;
```

 All rights reserved © Japan Smartphone Security Association.

```
import android.os.IBinder;
import android.widget.Toast;

public class PrivateStartService extends Service{
    // Service が起動するときに 1 回だけ呼び出される
    @Override
    public void onCreate() {
        Toast.makeText(this, this.getClass().getSimpleName() + " - 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 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。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param), Toast.LENGTH_LONG).show();

        // 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() {
        Toast.makeText(this, this.getClass().getSimpleName() + " - onDestroy()", Toast.LENGTH_SHORT).show();
    }

    @Override
    public IBinder onBind(Intent intent) {
        // このサービスにはバインドしない
        return null;
    }
}
```

次に非公開 Service を利用する Activity のサンプルコードを示す。

ポイント(Service を利用する)：

4. 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
5. 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
6. 結果を受け取る場合、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.service.privateservice;

import org.jssec.android.service.privateservice.R;

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

All rights reserved © Japan Smartphone Security Association.

```
public class PrivateUserActivity extends Activity {

    private Context mContext;

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

        setContentView(R.layout.privateservice_activity);

        mContext = this;
    }

    // サービス開始
    public void onStartServiceClick(View v) {
          // ★ポイント 4★ 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
        Intent intent = new Intent(mContext, PrivateStartService.class);

          // ★ポイント 5★ 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");

        startService(intent);
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doStopService();
    }

    // IntentService 開始ボタン

    public void onIntentServiceClick(View v) {
          // ★ポイント 4★ 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
        Intent intent = new Intent(mContext, PrivateIntentService.class);

          // ★ポイント 5★ 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");

        startService(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
        doStopService();
    }

    // サービスを停止する
    private void doStopService() {
        // ★ポイント 4★ 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
        Intent intent = new Intent(mContext, PrivateStartService.class);
        stopService(intent);
    }
}
```


All rights reserved © Japan Smartphone Security Association.

4.4.1.2. 公開 Service を作る・利用する

公開 Service は、不特定多数のアプリに利用されることを想定した Service である。マルウェアが送信した情報(Intent など)を受信することがあることに注意が必要である。
また、公開 Service を利用するには、送信する情報(Intent など)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。

以下、IntentService 型の Service を使用した例を示す。

ポイント(Service を作る):
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.service.publicservice"
    android:versionCode="1"
    android:versionName="1.0" >

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

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

        <!-- 最も標準的な Service -->
        <!-- 公開する場合は、exported=true にする -->
        <service android:name=".PublicStartService" android:exported="true">
            <intent-filter>
                <action android:name="org.jssec.android.service.publicservice.action.startservice" />
            </intent-filter>
        </service>

        <!-- IntentService を継承した Service -->
        <!-- 公開する場合は、exported=true にする -->
        <service android:name=".PublicIntentService" android:exported="true">
            <intent-filter>
                <action android:name="org.jssec.android.service.publicservice.action.intentservice" />
            </intent-filter>
        </service>

    </application>

</manifest>
```

PublicIntentService.java
```
package org.jssec.android.service.publicservice;

import android.app.IntentService;
import android.content.Intent;
import android.widget.Toast;
```

All rights reserved © Japan Smartphone Security Association.

```
public class PublicIntentService extends IntentService{

    /**
     * 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 PublicIntentService() {
        super("CreatingTypeBService");
    }

    // Service が起動するときに 1 回だけ呼び出される
    @Override
    public void onCreate() {
        super.onCreate();

        Toast.makeText(this, this.getClass().getSimpleName() + " - onCreate()", Toast.LENGTH_SHORT).show();
    }

    // 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★ 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。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param), Toast.LENGTH_LONG).show();
    }


    // Service が終了するときに 1 回だけ呼び出される
    @Override
    public void onDestroy() {
        Toast.makeText(this, this.getClass().getSimpleName() + " - onDestroy()", Toast.LENGTH_SHORT).show();
    }

}
```

次に公開 Service を利用する Activity のサンプルコードを示す。

ポイント(Service を利用する):

3. センシティブな情報を送信してはならない
4. 結果を受け取る場合、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.service.publicserviceuser"
    android:versionCode="1"
    android:versionName="1.0" >

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

    <application
        android:icon="@drawable/ic_launcher"
```

 All rights reserved © Japan Smartphone Security Association.

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

PublicUserActivity.java

```
package org.jssec.android.service.publicserviceuser;

import org.jssec.android.service.publicserviceuser.R;

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

public class PublicUserActivity extends Activity {

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

        setContentView(R.layout.publicservice_activity);
    }

    // サービス開始
    public void onStartServiceClick(View v) {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.startservice");

        // ★ポイント 3★ センシティブな情報を送信してはならない
        intent.putExtra("PARAM", "センシティブではない情報");

        startService(intent);
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doStopService();
    }

    // IntentService 開始ボタン

    public void onIntentServiceClick(View v) {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.intentservice");

        // ★ポイント 3★ センシティブな情報を送信してはならない
        intent.putExtra("PARAM", "センシティブではない情報");

        startService(intent);
    }
```

All rights reserved © Japan Smartphone Security Association.

```
    @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
        doStopService();
    }

    // サービスを停止する
    private void doStopService() {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.startservice");
        stopService(intent);
    }
}
```

 All rights reserved © Japan Smartphone Security Association.

4.4.1.3. パートナー限定 Service

パートナー限定 Service は、特定のアプリだけから利用できる Service 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。

以下、AIDL bind 型の Service を使用した例を示す。

ポイント(Service を作る):

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. onBind(onStartCommand,onHandleIntent)で呼び出し元がパートナーかどうか判別できない
4. 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
5. 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.service.exclusiveservice.aidl"
    android:versionCode="1"
    android:versionName="1.0" >

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

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

       <!-- AIDL を利用した Service -->
        <!-- ★ポイント 1★ Intent Filter を定義してはならない -->
       <service
           android:name=".ExclusiveAIDLService"
           android:exported="true" />
    </application>

</manifest>
```

今回の例では 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ーフェースで、もう1つは Activity から Service にデータを渡し、情報を取得するインターフェースである。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。

IExclusiveAIDLServiceCallback.aidl

```
package org.jssec.android.service.exclusiveservice.aidl;
```

All rights reserved © Japan Smartphone Security Association.

```
interface IExclusiveAIDLServiceCallback {
    /**
     * 値が変わった時に呼び出される
     */
    void valueChanged(String info);
}
```

IExclusiveAIDLService.aidl

```
package org.jssec.android.service.exclusiveservice.aidl;

import org.jssec.android.service.exclusiveservice.aidl.IExclusiveAIDLServiceCallback;

interface IExclusiveAIDLService {

    /**
     * コールバックを登録する
     */
    void registerCallback(IExclusiveAIDLServiceCallback cb);

    /**
     * 情報を取得する
     */
    String getInfo(String param);

    /**
     * コールバックを解除する
     */
    void unregisterCallback(IExclusiveAIDLServiceCallback cb);
}
```

ExclusiveAIDLService.java

```
package org.jssec.android.service.exclusiveservice.aidl;

import org.jssec.android.service.exclusiveservice.aidl.IExclusiveAIDLService;
import org.jssec.android.service.exclusiveservice.aidl.IExclusiveAIDLServiceCallback;
import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteCallbackList;
import android.os.RemoteException;
import android.widget.Toast;

public class ExclusiveAIDLService extends Service {
       private static final int REPORT_MSG = 1;
       private static final int GETINFO_MSG = 2;

    // Service からクライアントに通知する値
    private int mValue = 0;

    // ★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);
```

 All rights reserved © Japan Smartphone Security Association.

```
        sWhitelists = new PkgCertWhitelists();

        // パートナーアプリ org.jssec.android.service.exclusiveservice.aidluser の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.service.exclusiveservice.aidluser", 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);
    }

    // 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<IExclusiveAIDLServiceCallback> mCallbacks =
        new RemoteCallbackList<IExclusiveAIDLServiceCallback>();

    // 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++) {
                    IExclusiveAIDLServiceCallback target = mCallbacks.getBroadcastItem(i);
                    try {
                            // ★ポイント 5★ パートナーアプリに開示してよい情報に限り送信してよい
                        target.valueChanged("パートナーアプリに開示してよい情報(callback from Service) No." + (++
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();

                // 10 秒後に繰り返す
                sendEmptyMessageDelayed(REPORT_MSG, 10000);
                break;
             }
            case GETINFO_MSG: {
                Toast.makeText(ExclusiveAIDLService.this,
                       (String)msg.obj, Toast.LENGTH_LONG).show();
                break;
              }
            default:
                super.handleMessage(msg);
                break;
            } // switch
        }
    };
```

All rights reserved © Japan Smartphone Security Association.

```
// AIDL で定義したインターフェース
private final IExclusiveAIDLService.Stub mBinder = new IExclusiveAIDLService.Stub() {
    private final boolean checkPartner() {
        Context ctx = ExclusiveAIDLService.this;
        if (!ExclusiveAIDLService.checkPartner(ctx, Utils.getPackageNameFromPid(ctx, getCallingPid()))) {
            Toast.makeText(ctx, "利用元アプリはパートナーアプリではない。", Toast.LENGTH_LONG).show();
            return false;
        }
       return true;
    }
    public void registerCallback(IExclusiveAIDLServiceCallback cb) {
        // ★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()) {
            return;
        }
      if (cb != null) mCallbacks.register(cb);
    }
    public String getInfo(String param) {
        // ★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()) {
            return null;
        }
        // ★ポイント 4★ 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。「3.2 入力データの安全性を確認する」を参照。
      Message msg = new Message();
      msg.what = GETINFO_MSG;
      msg.obj = String.format("パートナーアプリからのメソッド呼び出し。「%s」を受信した。", param);
      ExclusiveAIDLService.this.mHandler.sendMessage(msg);

        // ★ポイント 5★ 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
        return new String("パートナーアプリに開示してよい情報(method from Service)");
    }

    public void unregisterCallback(IExclusiveAIDLServiceCallback cb) {
        // ★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()) {
            return;
        }

       if (cb != null) mCallbacks.unregister(cb);
    }
};

@Override
public IBinder onBind(Intent intent) {
    // ★ポイント 3★ onBind で呼び出し元がパートナーかどうか判別できない
    // AIDL で定義したメソッドの呼び出し毎にチェックが必要になる。

    return mBinder;
}

@Override
public void onCreate() {
    Toast.makeText(this, this.getClass().getSimpleName() + " - 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);
}
```

 All rights reserved © Japan Smartphone Security Association.

```
    @Override
    public void onDestroy() {
        Toast.makeText(this, this.getClass().getSimpleName() + " - onDestroy()", Toast.LENGTH_SHORT).show();

        // コールバックを全て解除する
        mCallbacks.kill();

        mHandler.removeMessages(REPORT_MSG);
    }
}
```

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.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

次にパートナー限定 Service を利用する Activity のサンプルコードを示す。

ポイント(Service を利用する)：

6. 利用先パートナー限定 Service 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
7. 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
8. 明示的 Intent によりパートナー限定 Service を呼び出す
9. 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

ExclusiveAIDLUserActivity.java

```
package org.jssec.android.service.exclusiveservice.aidluser;
```


All rights reserved © Japan Smartphone Security Association.

```
import org.jssec.android.service.exclusiveservice.aidl.IExclusiveAIDLService;
import org.jssec.android.service.exclusiveservice.aidl.IExclusiveAIDLServiceCallback;
import org.jssec.android.service.exclusiveservice.aidluser.R;
import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

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

public class ExclusiveAIDLUserActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    private final static int MGS_VALUE_CHANGED = 1;

    // ★ポイント 6★ 利用先パートナー限定 Service 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 Service アプリ org.jssec.android.service.exclusiveservice.aidl の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.service.exclusiveservice.aidl", 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 Service 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.service.exclusiveservice.aidl";
    private static final String TARGET_CLASS = "org.jssec.android.service.exclusiveservice.aidl.ExclusiveAIDLServ
ice";

    private final Handler mHandler = new Handler() {
        @Override public void handleMessage(Message msg) {
            switch (msg.what) {
                case MGS_VALUE_CHANGED: {
                      String info = (String)msg.obj;
                    Toast.makeText(mContext, String.format("コールバックで「%s」を受信した。", info), Toast.LENGT
H_SHORT).show();
                    break;
```

All rights reserved © Japan Smartphone Security Association.

```
                }
            default:
                super.handleMessage(msg);
                break;
            } // switch
        }
    };

    // AIDL で定義したインターフェース。Service からの通知を受け取る。
    private final IExclusiveAIDLServiceCallback.Stub mCallback =
        new IExclusiveAIDLServiceCallback.Stub() {
            @Override
            public void valueChanged(String info) throws RemoteException {
                Message msg = mHandler.obtainMessage(MGS_VALUE_CHANGED, info);
                mHandler.sendMessage(msg);
            }
    };

    // AIDL で定義したインターフェース。Service へ通知する。
    private IExclusiveAIDLService 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 = IExclusiveAIDLService.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.exclusiveservice_activity);

        mContext = this;
    }

    // サービス開始ボタン
    public void onStartServiceClick(View v) {
        // bindService を実行する
        doBindService();
```

 All rights reserved © Japan Smartphone Security Association.

```
    }

    // 情報取得ボタン
    public void onGetInfoClick(View v) {
        getServiceinfo();
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doUnbindService();
    }

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

    /**
     * Service に接続する
     */
    private void doBindService() {
        if (!mIsBound){
            // ★ポイント 6★ 利用先パートナー限定 Service 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 Service アプリはホワイトリストに登録されていない。", Toast.LENGTH_LO
NG).show();
                return;
            }

            Intent intent = new Intent();

            // ★ポイント 7★ 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 8★ 明示的 Intent によりパートナー限定 Service を呼び出す
            intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

          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);
```

All rights reserved © Japan Smartphone Security Association.

```
            Intent intent = new Intent();

            // ★ポイント 8★ 明示的 Intent によりパートナー限定 Service を呼び出す
            intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

            stopService(intent);

            mIsBound = false;
        }
    }

    /**
     * Service から情報を取得する
     */
    void getServiceinfo() {
        if (mIsBound && mService != null) {
            String info = null;

            try {
                // ★ポイント 7★ 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
                info = mService.getInfo(new String("パートナーアプリに開示してよい情報(method from activity)"));
            } catch (RemoteException e) {
            }
            // ★ポイント 9★ 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。「3.2 入力データの安全性を確認する」を参照。
            Toast.makeText(mContext, String.format("サービスから「%s」を取得した。", info), Toast.LENGTH_SHORT).s
how();
        }
    }
}
```

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
```


All rights reserved © Japan Smartphone Security Association.

```
        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.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
```

All rights reserved © Japan Smartphone Security Association.

```
}
```

 All rights reserved © Japan Smartphone Security Association.

4.4.1.4. 自社限定 Service

自社限定 Service は、自社以外のアプリから利用されることを禁止する Service 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。

以下、Messenger bind 型の Service を使用した例を示す。

ポイント(Service を作る):

1. 独自定義 Signature Permission を定義する
2. Service 定義にて、独自定義 Signature Permission を要求宣言する
3. Service 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
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.proprietaryservice.messenger"
    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.service.proprietaryservice.messenger.MY_PERMISSION"
        android:protectionLevel="signature" />

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

        <!-- Messenger を利用した Service -->
         <!-- ★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 -->
        <service
            android:name=".ProprietaryMessengerService"
            android:exported="true"
            android:permission="org.jssec.android.service.proprietaryservice.messenger.MY_PERMISSION" />
    </application>

</manifest>
```

ProprietaryMessengerService.java

```java
package org.jssec.android.service.proprietaryservice.messenger;

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

All rights reserved © Japan Smartphone Security Association.

```
import java.util.ArrayList;
import java.util.Iterator;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
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.widget.Toast;

public class ProprietaryMessengerService extends Service{
    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.service.proprietaryservice.messenger.MY_PERMIS
SION";

    // 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;
    }


    // 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 CommonValue.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 CommonValue.MSG_UNREGISTER_CLIENT:
                mClients.remove(msg.replyTo);
                break;
            case CommonValue.MSG_SET_VALUE:
                // クライアントにデータを送る
                sendMessageToClients();
                break;
            default:
                super.handleMessage(msg);
```

 All rights reserved © Japan Smartphone Security Association.

```
                break;
            }
        }
    }

    /**
     * クライアントにデータを送る
     */
    private void sendMessageToClients(){

        // ★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
        String sendValue = "センシティブな情報(from 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 sendMsg = Message.obtain(null, CommonValue.MSG_SET_VALUE, null);

                Bundle data = new Bundle();
                data.putString("key", sendValue);
                sendMsg.setData(data);

                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 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();
            return null;
        }

        // ★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。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param), Toast.LENGTH_LONG).show();

        return mMessenger.getBinder();
    }

    @Override
    public void onCreate() {
        Toast.makeText(this, "Service - onCreate()", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "Service - onDestroy()", Toast.LENGTH_SHORT).show();
```

All rights reserved © Japan Smartphone Security Association.

```
    }
}
```

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;
        }
    }
}
```

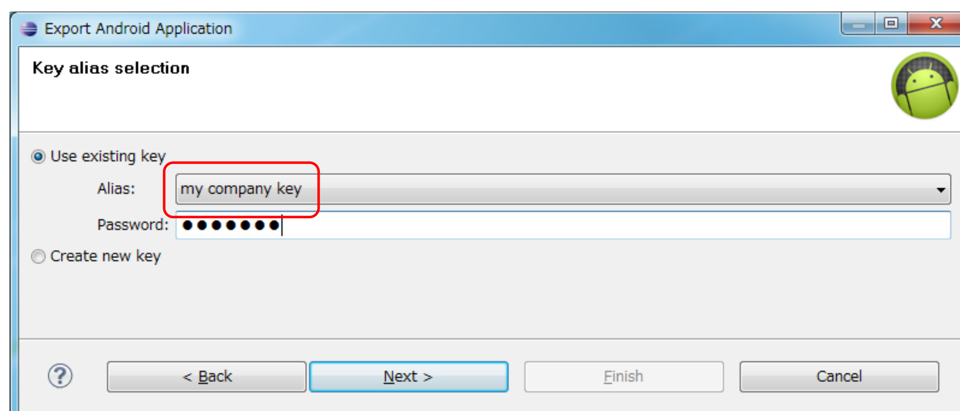
PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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(" ", "");
```

 All rights reserved © Japan Smartphone Security Association.

```
        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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.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。



次に自社限定 Service を利用する Activity のサンプルコードを示す。

ポイント(Service を利用する):

8. 独自定義 Signature Permission を利用宣言する
9. 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
10. 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
11. 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
12. 明示的 Intent により自社限定 Service を呼び出す
13. 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
14. 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.service.proprietaryservice.messengeruser"
    android:versionCode="1"
    android:versionName="1.0" >

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

    <!-- ★ポイント 8★ 独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.service.proprietaryservice.messenger.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name="org.jssec.android.service.proprietaryservice.messengeruser.ProprietaryMessengerUserActi
vity"
            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>
```

ProprietaryMessengerUserActivity.java

```
package org.jssec.android.service.proprietaryservice.messengeruser;

import org.jssec.android.service.proprietaryservice.messengeruser.R;
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.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;
```

 All rights reserved © Japan Smartphone Security Association.

```
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.view.View;
import android.widget.Toast;

public class ProprietaryMessengerUserActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    // 利用先の Activity 情報
    private static final String TARGET_PACKAGE =  "org.jssec.android.service.proprietaryservice.messenger";
    private static final String TARGET_CLASS = "org.jssec.android.service.proprietaryservice.messenger.Proprietar
yMessengerService";

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.service.proprietaryservice.messenger.MY_PERMIS
SION";

    // 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;
    }

    // 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 CommonValue.MSG_SET_VALUE:
                    Bundle data = msg.getData();
                    String info = data.getString("key");
                       // ★ポイント 13★ 自社アプリからの結果情報であっても、値の安全性を確認する
                       // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
                    Toast.makeText(mContext, String.format("サービスから「%s」を取得した。", info),
                            Toast.LENGTH_SHORT).show();
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }
```

All rights reserved © Japan Smartphone Security Association.

```
    // 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, CommonValue.MSG_REGISTER_CLIENT);
                msg.replyTo = mActivityMessenger;
                mServiceMessenger.send(msg);
            } catch (RemoteException e) {
                // Service が異常終了していた場合
            }
        }

        // 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) {
            mServiceMessenger = null;
            Toast.makeText(mContext, "Disconnected from service", Toast.LENGTH_SHORT).show();
        }
    };

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

        setContentView(R.layout.publicservice_activity);

        mContext = this;
    }
    // サービス開始ボタン
    public void onStartServiceClick(View v) {
        // bindService を実行する
        doBindService();
    }

    // 情報取得ボタン
    public void onGetInfoClick(View v) {
        getServiceinfo();
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doUnbindService();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        doUnbindService();
    }

    /**
     * Service に接続する
     */
```

 All rights reserved © Japan Smartphone Security Association.

```
    void doBindService() {
        if (!mIsBound){
            // ★ポイント 9★ 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.L
ENGTH_LONG).show();
                return;
            }

            // ★ポイント 10★ 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, "利用先サービスは自社アプリではない。", Toast.LENGTH_LONG).show();
                return;
            }

          Intent intent = new Intent();

            // ★ポイント 11★ 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 12★ 明示的 Intent により自社限定 Service を呼び出す
          intent.setClassName(TARGET_PACKAGE, TARGET_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;
        }
    }

    /**
     * Service から情報を取得する
     */
    void getServiceinfo() {
        if (mServiceMessenger != null) {
            try {
                  // 情報の送信を要求する
                Message msg = Message.obtain(null, CommonValue.MSG_SET_VALUE);
                mServiceMessenger.send(msg);
            } catch (RemoteException e) {
                // Service が異常終了していた場合
            }
         }
    }

}
```

SigPerm.java

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

All rights reserved © Japan Smartphone Security Association.

```
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;
        }
    }
}
```

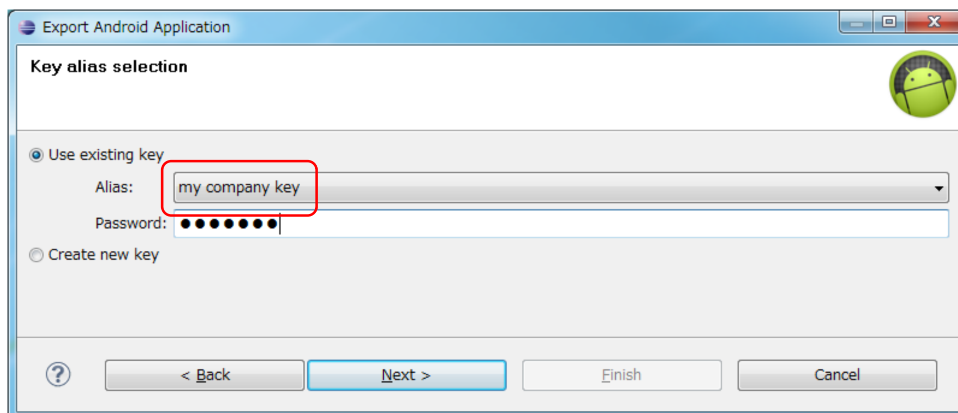
PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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();
```

 All rights reserved © Japan Smartphone Security Association.

```
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

★ポイント 14★ 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。



All rights reserved © Japan Smartphone Security Association.

4.4.2. ルールブック

Service 実装時には以下のルールを守ること。

1.	アプリ内でのみ使用する Service は非公開設定する	（必須）
2.	受信データ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.	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）
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）
6.	利用先 Service が固定できる場合は明示的 Intent で Service を利用する	（必須）
7.	他社の特定アプリと連携する場合は利用先 Service を確認する	（必須）
8.	資産を二次的に提供する場合には、その資産の従来の保護水準を維持する	（必須）
9.	センシティブな情報はできる限り送らない	（推奨）

4.4.2.1. アプリ内でのみ使用する 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。

実装上は AndroidManifest.xml で Service を定義する際に、exported 属性を false にするだけである。

AndroidManifest.xml
```
        <!-- 非公開 Service -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <service android:name=".PrivateStartService" android:exported="false"/>
```

また、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される Service であり、かつ Intent Filter を設置するような設計はしてはならない。Intent Filter の性質上、同一アプリ内の非公開 Service を呼び出すつもりでも、Intent Filter 経由で呼び出したときに意図せず他アプリの公開 Service を呼び出してしまう場合が存在するからである。

AndroidManifest.xml(非推奨)
```
        <!-- 非公開 Service -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <service android:name=".PrivateStartService" android:exported="false">
            <intent-filter>
                <action android:name="org.jssec.android.service.OPEN />
            </intent-filter>
        </service>
```

「4.4.3.1 exported 設定と intent-filter 設定の組み合わせ(Service の場合)」も参照すること。

 All rights reserved © Japan Smartphone Security Association.

4.4.2.2. 受信データの安全性を確認する （必須）

Service も Activity と同様に、受信 Intent のデータを処理する際には、まず受信 Intent の安全性を確認しなければならない。Service を利用する側も Service からの結果(として受信した）情報の安全性を確認する必要がある。Activity の「4.1.2.5 受信 Intent の安全性を確認する （必須）」「4.1.2.9 利用先 Activity からの戻り Intent の安全性を確認する （必須）」も参照すること。

Service においては、Intent 以外にもメソッドの呼び出しや Message によるデータの送受信などがあるため、それぞれ注意して実装を行わなければならない。

「3.2 入力データの安全性を確認する」を参照すること。

4.4.2.3. 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 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。

4.4.2.4. 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。

4.4.2.5. 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）

Service のタイプによって結果情報の返送先（コールバックの呼び出し先や Message の送信先）アプリの信用度が異なる。返送先がマルウェアである可能性も考慮して十分に情報漏洩に対する配慮をしなければならない。

詳細は、Activity の「4.1.2.7 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）」を参照すること。

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

暗黙的 Intent により Service を利用すると、Intent Filter の定義が同じ場合には先にインストールした Service に Intent が送信されてしまう。もし意図的に同じ Intent Filter を定義したマルウェアが先にインストールされていた場合、

All rights reserved © Japan Smartphone Security Association.

マルウェアに Intent が送信されてしまい、情報漏洩が生じる。一方、明示的 Intent により Service を利用すると、指定した Service 以外が Intent を受信することはなく比較的安全である。

ただし、別途考慮すべき点があるので、Activity の「4.1.2.8 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）」を参照すること。

4.4.2.7. 他社の特定アプリと連携する場合は利用先 Service 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「4.4.1.3 パートナー限定 Service」を参照すること。

4.4.2.8. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する （必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4 Permission の再委譲問題」を参照すること。

4.4.2.9. 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。

センシティブな情報を Service と受け渡しする場合、その情報の漏洩リスクを検討しなければならない。公開 Service 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。またパートナー限定 Service や自社限定 Service に送付した情報もそれら Service の実装に依存して情報漏洩リスクの大小がある。

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 Service は信頼できる Service に限定し、情報が LogCat などに漏洩しないように配慮しなければならない。

All rights reserved © Japan Smartphone Security Association.

4.4.3. アドバンスト

4.4.3.1. exported 設定と intent-filter 設定の組み合わせ(Service の場合)

このガイド文書では、Service の用途から非公開 Service、公開 Service、パートナー限定 Service、自社限定 Service の 4 タイプの Service 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 Service 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	（使用禁止）	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

「intent-filter 定義がある」&「exported=false」を使用禁止にしているのは、Android の振る舞いとして、同一アプリ内の非公開 Service を呼び出したつもりでも、意図せず他アプリの公開 Service を呼び出してしまう場合が存在するためである。

具体的には、Android は以下のような振る舞いをするのでアプリ設計時に検討が必要である。

- 複数の Service で同じ内容の intent-filter を定義した場合、先にインストールしたアプリ内の Service の定義が優先される
- 暗黙的 Intent を使った場合は、OS によって優先の Service が自動的に選ばれて、呼び出される。

以下の 3 つの図で Android の振る舞いによる意図せぬ呼び出しが起こる仕組みを説明する。
図 4.4-1 は、同一アプリ内からしか非公開 Service(アプリ A)を暗黙的 Intent で呼び出せない正常な動作の例である。Intent-filter(図中 action="X")を定義しているのが、アプリ A しかいないので意図通りの動きとなっている。

All rights reserved © Japan Smartphone Security Association.

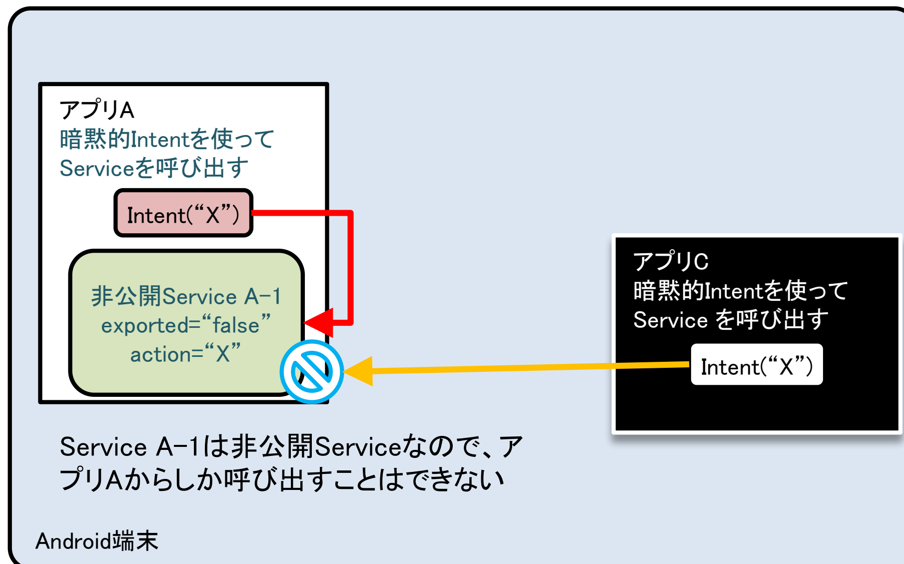


図 4.4-1

図 4.4-2 および図 4.4-3 は、アプリ A に加えてアプリ B でも同じ intent-filter(図中 action="X")を定義している場合である。

図 4.4-2 は、アプリ A→アプリ B の順でインストールされた場合である。この場合、アプリ C が暗黙的 Intent を送信すると、非公開の Service(A-1)を呼び出そうとして失敗する。一方、アプリ A は暗黙的 Intent を使って意図通りに同一アプリ内の非公開 Service を呼び出せるので、セキュリティの(マルウェア対策の)面では問題は起こらない。

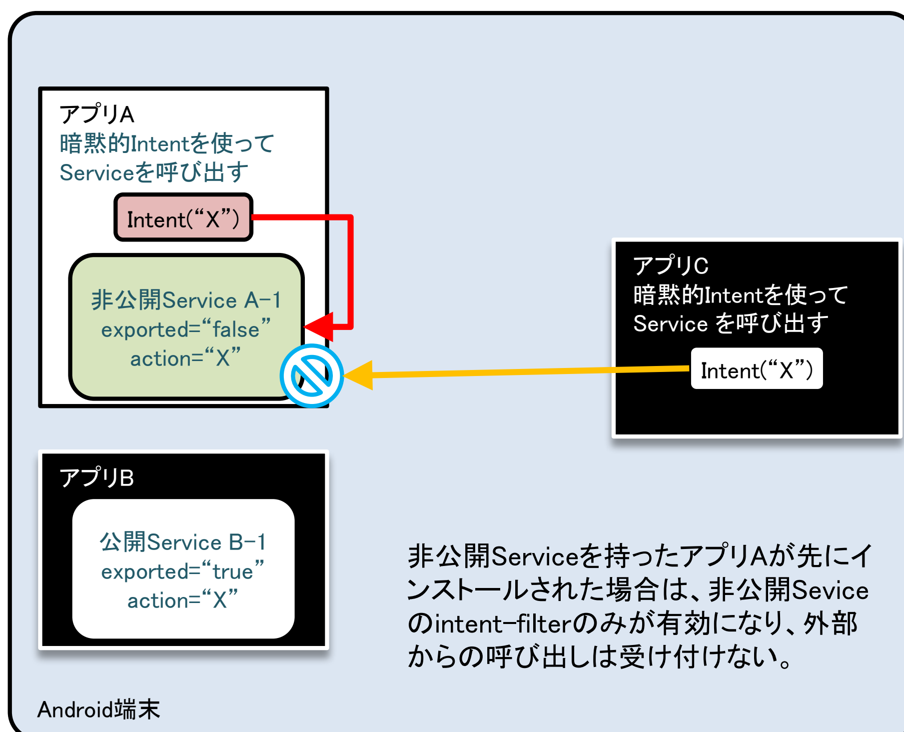


図 4.4-2

図 4.4-3 は、アプリ B→アプリ A の順でインストールされた場合であり、セキュリティ面からみて問題がある。アプリ A が暗黙的 Intent を送信して同一アプリ内の非公開 Service を呼び出そうとするが、先にインストールしたアプリ B の公開 Activity(B-1)が呼び出されてしまう例を示している。これによりアプリ A からアプリ B に対してセンシティブな情

 All rights reserved © Japan Smartphone Security Association.

報を送信する可能性が生じてしまう。アプリ B がマルウェアであれば、そのままセンシティブな情報の漏洩に繋がる。

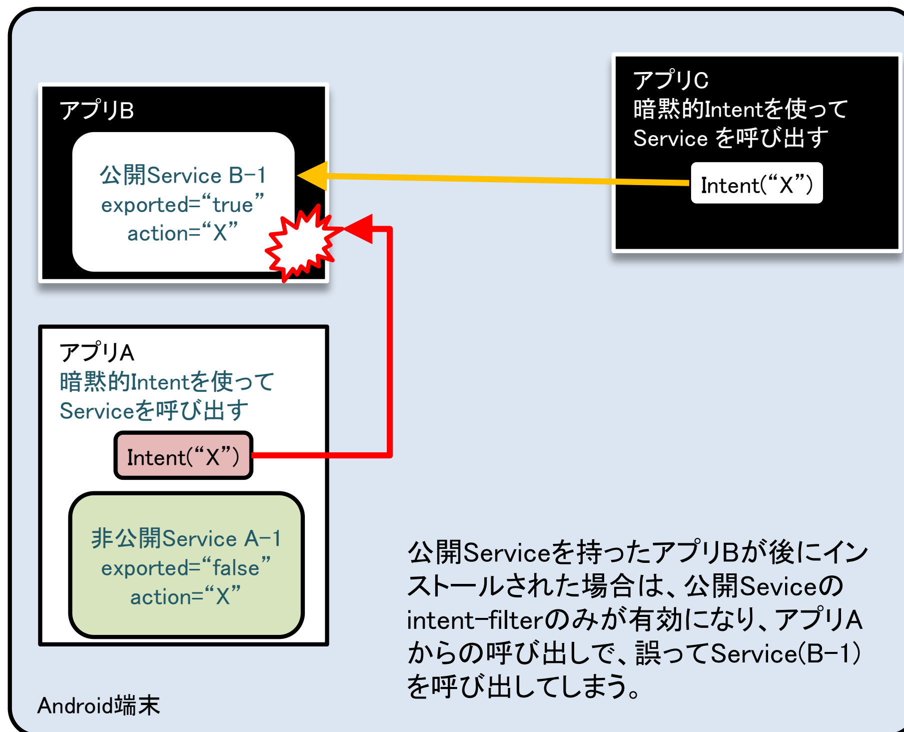


図 4.4-3

このように、Intent Filter を用いた非公開 Service の暗黙的 Intent 呼び出しは、意図せぬアプリの呼び出しや意図せぬアプリへのセンシティブな情報の送信を避けるためにも行うべきではない。

4.4.3.2. Service の実装方法について

Service の実装方法は多様であり、サンプルコードで分類したセキュリティ上のタイプとの相性もあるため簡単に特徴を示す。

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が、startService と bindService の両方で利用できる Service を作成することも可能である。

Service の実装方法を決定するために、次のような項目について検討を行うことになる。

- Service を別アプリに公開するか（Service の公開）
- 実行中にデータのやり取りを行うか(データの相互送受信)
- Service を制御するか（起動や終了など）
- 別プロセスとして実行するか（プロセス間通信）
- 複数の処理を同時に行うか(並行処理)

実装方法の分類と各々の項目の実現の可否を表にすると、表 4.4.3-1 のようになる。×は実現不可能かもしくは提

All rights reserved © Japan Smartphone Security Association.

供される機能とは別の枠組みが必要な場合を表す。

表 4.4.3-1 Service の実装方法の分類

分類	Service の公開	データの相互送受信	Service の制御 (起動・終了)	プロセス間通信	並行処理
startService 型	○	×	○	○	×
IntentService 型	○	×	×	○	×
local bind 型	×	○	○	×	×
Messenger bind 型	○	○	○	○	×
AIDL bind 型	○	○	○	○	○

startService 型

最も基本的な Service である。Service クラスを継承し、onStartCommand で処理を行う Service のことを指す。利用する側は、Service を Intent で指定して startService を使用して呼び出す。

Intent の送信元に対して、結果などのデータを直接返すことはできないため、Broadcast など別の方法を組み合わせて実現する必要がある。

具体的な実装例は、「4.4.1.1 非公開 Service を作る・利用する」を参照のこと。

セキュリティ上のチェックは onStartCommand で行う必要があるが、送信元のパッケージ名が取得できないためパートナー限定 Service には使用できない。

IntentService 型

IntentService は Service を継承して作られているクラスである。呼び出し方は、startService 型と同様である。

通常の Service(startService 型)に比べて以下の特徴がある。

- Intent の処理は onHandleIntent で行う。(onStartCommand は使わない)
- 別スレッドで実行される
- 処理がキューイングされる

処理が別スレッドのため呼び出しは即座に返され、キューイング機構によりシーケンシャルに Intent に対する処理が行われる。各 Intent の並行処理はされないが、製品の要件によっては実装の簡素化の一つとして選択が可能である。

Intent の送信元に対して、結果などのデータを直接返すことはできないため、Broadcast など別の方法を組み合わせて実現する必要がある。

具体的な実装例は、「4.4.1.2 公開 Service を作る・利用する」を参照のこと。

セキュリティ上のチェックは onHandleIntent で行う必要があるが、送信元のパッケージ名が取得できないためパートナー限定 Service には使用できない。

local bind 型

アプリと同じプロセス内でのみ動くローカル Service を実装するための方法を指す。
Binder クラスから派生したクラスを定義して、Service で実装した機能(メソッド)を呼び出し元に提供できるようにする。
利用する側は、Service を Intent で指定して bindService を使用して呼び出す。

Service を bind する方法の中では、最もシンプルな実装であるが、別プロセスでの起動や Service の公開ができないため用途は限定される。

具体的な実装例は、サンプルコードに含まれるプロジェクト「Service PrivateServiceLocalBind」を参照のこと。

セキュリティ的には非公開 Service のみ実装可能である。

Messenger bind 型

Messenger の仕組みを利用して Service との連携を実現する方法を指す。
Service を利用する側からも Message の返信先として Messenger を渡すことができるため、双方でのデータのやり取りが比較的容易に実現可能である。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。各 Message の並行処理はされないが、製品の要件によっては実装の簡素化の一つとして選択が可能である。
利用する側は、Service を Intent で指定して bindService を使用して呼び出す。

具体的な実装例は、「4.4.1.4 自社限定 Service」を参照のこと。

セキュリティ上のチェックは onBind や Message Handler で行う必要があるが、送信元のパッケージ名が取得できないためパートナー限定 Service には使用できない。

AIDL bind 型

AIDL の仕組みを利用して Service との連携を実現する方法を指す。
AIDL によってインターフェースを定義し、Service の持つ機能をメソッドとして提供する。また、AIDL で定義したインターフェースを利用側で実装することで、コールバックを実現することもできる。
マルチスレッド呼び出しは可能だが、排他処理はされないので Service 側で明示的に実装する必要がある。

利用する側は、Service を Intent で指定して bindService を使用して呼び出す。

All rights reserved © Japan Smartphone Security Association.

具体的な実装例は、「4.4.1.3 パートナー限定 Service」を参照のこと。

セキュリティ上のチェックは自社限定 Service では onBind で、パートナー限定 Service では AIDL で定義したインターフェースの各メソッドで行う必要がある。本文書で分類した全セキュリティタイプの Service に利用可能である。

 All rights reserved © Japan Smartphone Security Association.

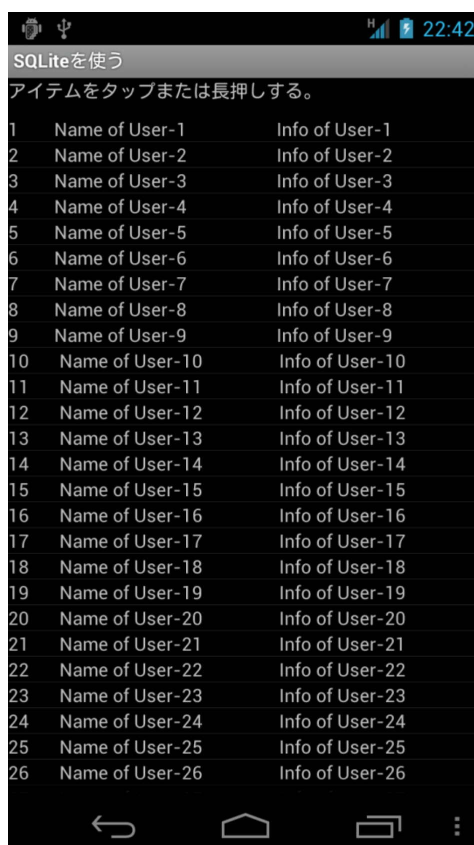
4.5. SQLite を使う

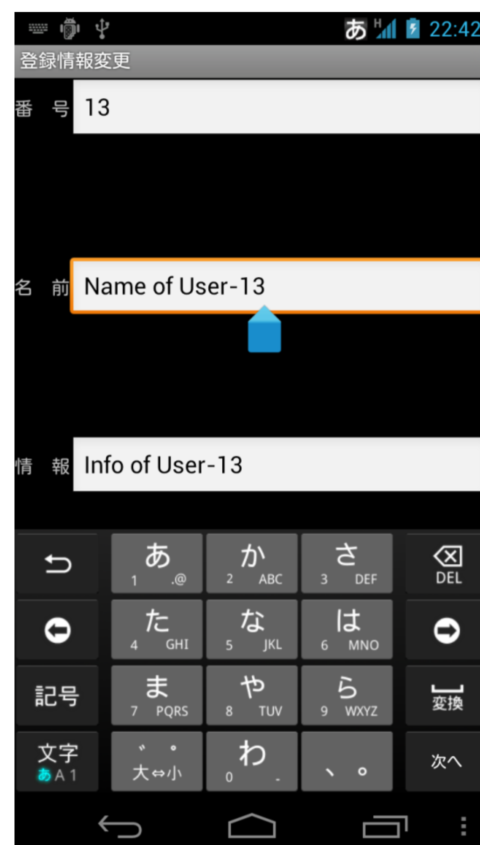
本文書では SQLite を使用してデータベースの作成および操作を行う際にセキュリティ上で注意すべき点をまとめる。主なポイントは、データベースファイルのアクセス権の適切な設定と SQL インジェクションに対する対策である。ここでは、直接外部からデータベースファイルの読み書きを許す(複数アプリで共有する)ようなデータベースはここでは想定せず Content Provider のバックエンドやアプリ単体での使用を前提とする。また、ある程度センシティブな情報を扱っていることを想定しているが、そうでない場合も他アプリからの想定外の読み書きを避けるためにもここで挙げる対策を適用することをお勧めする。

4.5.1. サンプルコード

4.5.1.1. 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[4]。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。





[4] 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。そのため、誤って SD カード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ので注意が必要である。

All rights reserved © Japan Smartphone Security Association.

ポイント:

1. データベース作成には SQLiteOpenHelper を使用する
2. 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
3. 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) {
```


All rights reserved © Japan Smartphone Security Association.

```
            //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 UNIQUE, "
            + "name VARCHAR(" + CommonData.TEXT_DATA_LENGTH_MAX + ") NOT NULL, "
            + "info VARCHAR(" + CommonData.TEXT_DATA_LENGTH_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) {
        // 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;
```

All rights reserved © Japan Smartphone Security Association.

```
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;
            }
```

All rights reserved © Japan Smartphone Security Association.

```
        } 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_LENGTH_MAX)) {
            return false;
        }
        if (!validateLength(info, CommonData.TEXT_DATA_LENGTH_MAX)) {
            return false;
        }
        return true;
    }
}
```

 All rights reserved © Japan Smartphone Security Association.

4.5.2. 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）

4.5.2.1. DB ファイルの配置場所、アクセス権を正しく設定する （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、アプリディレクトリに配置した場合でも、アクセス権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設定については、DB 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 で取得できるディレクトリの場所に配置する[5]
2. アクセス権
   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。

これらを実施するためには以下の方法が挙げられる。

1. SQLiteOpenHelper を使用する
2. Context#openOrCreateDatabase を使用する

DB ファイルの作成に際しては、SQLiteDatabase#openOrCreateDatabase を使用することもできる。しかし、このメソッドを使用した場合、Android スマートフォンの機種によっては、他のアプリから読み取り可能な DB ファイルが作成されることが分っている。そのため、このメソッドの使用は避けて、他の方法を利用することを推奨する。

[5] どちらのメソッド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ディレクトリ（パッケージディレクトリ）のサブディレクトリ以下のパスが取得できる。

All rights reserved © Japan Smartphone Security Association.

上に挙げた 2 つの方法について、それぞれの特徴を以下で説明する。

SQLiteOpenHelper を使用する

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 DB の名前(ファイル名に使われる)[6]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。

「4.5.1.1 データベースの作成と操作」に具体的な使用方法を示しているので参照すること。

Context#openOrCreateDatabase を使用する

SQLiteDatabase#openOrCreateDatabase メソッドを使用して DB の作成を行う場合、ファイルのアクセス権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ため注意が必要である。

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

6 (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。

 All rights reserved © Japan Smartphone Security Association.

要件に照らし合わせて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

4.5.2.2. 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。

4.5.2.3. 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。

All rights reserved © Japan Smartphone Security Association.

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;

    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];

        //★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;
        }
        //データ追加処理
        //プレースホルダを使用する
        String commandString = "INSERT INTO " + CommonData.TABLE_NAME + " (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) {
            Log.e(DataInsertTask.class.toString(), mActivity.getString(R.string.UPDATING_ERROR_MESSAGE));
        } finally {
               sqlStmt.close();
        }
        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。

All rights reserved © Japan Smartphone Security Association.

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。

All rights reserved © Japan Smartphone Security Association.

4.5.3. アドバンスト

4.5.3.1. 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。

実際のエスケープ処理は、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);
```

 All rights reserved © Japan Smartphone Security Association.

```
    }
}
```

4.5.3.2. 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

4.5.3.3. 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[7]。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 ・・・
```

[7] 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()

4.5.3.4. アプリの要件に従って 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コードを示す。

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
```


All rights reserved © Japan Smartphone Security Association.

```
                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 ・・・
}
```

4.5.3.5. 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、root 権限を奪取されるなど 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、端末の root 権限が奪取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。

参考として「4.5.3.6 [参考]SQLite データベースを暗号化する(SQLCipher for Android)」に、データベースを暗号化するライブラリを紹介しておく。

All rights reserved © Japan Smartphone Security Association.

4.5.3.6. [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

 All rights reserved © Japan Smartphone Security Association.

- 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。

All rights reserved © Japan Smartphone Security Association.

4.6. ファイルを扱う

Android のセキュリティ設計思想に従うと、ファイルは情報を永続化又は一時保存(キャッシュ)する目的にのみ利用し、原則非公開にするべきである。アプリ間の情報交換はファイルを直接アクセスさせるのではなく、ファイル内の情報をContent ProviderやServiceといったアプリ間連携の仕組みによって交換するべきである。これによりアプリ間のアクセス制御も実現できる。

SD カード等の外部記憶デバイスは十分なアクセス制御ができないため、容量の大きなファイルを扱う場合や別の場所(PC など)への情報の移動目的など、機能上どうしても必要な場合のみに使用を限定するべきである。基本的に外部記憶デバイス上にはセンシティブな情報を含んだファイルを配置してはならない。もしセンシティブな情報を外部記憶デバイス上のファイルに保存しなければならない場合は暗号化等の対策が必要になるが、ここでは言及しない。

4.6.1. サンプルコード

前述のようにファイルは原則非公開にするべきである。しかしながらさまざまな事情によって、他のアプリにファイルを直接読み書きさせるべきときもある。セキュリティの観点から分類したファイルの種類と比較を表 4.6.1-1 に示す。ファイルの格納場所や他アプリへのアクセス許可の組み合わせにより 4 種類のファイルに分類している。以降ではこのファイルの分類ごとにサンプルコードを示し説明を加えていく。

表 4.6.1-1 セキュリティ観点によるファイルの分類と比較

ファイルの分類	他アプリへのアクセス許可	格納場所	概要
非公開ファイル	なし	アプリディレクトリ内	● アプリ内でのみ読み書きできる。 ● センシティブな情報を扱うことができる。 ● ファイルは原則このタイプにするべき。
読み取り公開ファイル	読み取り	アプリディレクトリ内	● 他アプリおよびユーザーも読み取り可能。 ● アプリ外部に公開(閲覧)可能な情報を扱う。
読み書き公開ファイル	読み取り 書き込み	アプリディレクトリ内	● 他アプリおよびユーザーも読み書き可能。 ● セキュリティの観点からもアプリ設計の観点からも使用は避けるべき。
外部記憶ファイル (読み書き公開)	読み取り 書き込み	SD カードなどの外部記憶装置	● アクセス権のコントロールができない。 ● 他アプリやユーザーによるファイルの読み書き・削除が常に可能。 ● 使用は必要最小限にするべき。 ● 比較的容量の大きなファイルを扱うことができる。

 All rights reserved © Japan Smartphone Security Association.

4.6.1.1. 非公開ファイルを扱う

同一アプリ内でのみ読み書きされるファイルを扱う場合であり、安全なファイルの使い方である。ファイルに格納する情報が公開可能かどうかに関わらず、できるだけファイルは非公開の状態で保持し、他アプリとの必要な情報のやり取りは別の Android の仕組み（Content Provider、Service)を利用して行うことを原則とする。

ポイント:

1. ファイルは、アプリディレクトリ内に作成する
2. ファイルのアクセス権は、他のアプリが利用できないようにプライベートモードにする
3. センシティブな情報を格納することができる
4. ファイルに格納する(された)情報に対しては、その入手先に関わらず内容の安全性を確認する

PrivateFileActivity.java

```
package org.jssec.android.file.privatefile;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.privatefile.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateFileActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "private_file.dat";

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

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * ファイルの作成処理
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント 1★ ファイルは、アプリディレクトリ内に作成する
            // ★ポイント 2★ ファイルのアクセス限は、他のアプリが利用できないようにプライベートモードにする
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);
```

All rights reserved © Japan Smartphone Security Association.

```
            // ★ポイント 3★ センシティブな情報を格納することができる
            // ★ポイント 4★ ファイルに格納する情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            fos.write(new String("センシティブな情報(File Activity)¥n").getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
        }

        finish();
    }

    /**
     * ファイルの読み込み処理
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * ファイルの削除処理
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(this.getFilesDir() + "/" + FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
```


All rights reserved © Japan Smartphone Security Association.

```
    }
}
```

PrivateUserActivity.java

```
package org.jssec.android.file.privatefile;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.privatefile.R;

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

public class PrivateUserActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "private_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClass(this, PrivateFileActivity.class);

        startActivity(intent);
    }

    /**
     * ファイル Activity の呼び出し処理
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * ファイルの読み込み処理
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);
```

All rights reserved © Japan Smartphone Security Association.

```
            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // ★ポイント 4★ ファイルに格納された情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * ファイルの追記処理
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント 1★ ファイルは、アプリケーションディレクトリ内に作成する
            // ★ポイント 2★ ファイルのアクセス限は、他のアプリが利用できないようにプライベートモードにする
            fos = openFileOutput(FILE_NAME, MODE_APPEND);

            // ★ポイント 3★ センシティブな情報を格納することができる
            // ★ポイント 4★ ファイルに格納する情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            fos.write(new String("センシティブな情報(User Activity)¥n").getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
        }

        callFileActivity();
    }

}
```

 All rights reserved © Japan Smartphone Security Association.

4.6.1.2. 読み取り公開ファイルを扱う

不特定多数のアプリに対して内容を公開するためのファイルである。以下のポイントに気を付けて実装すれば、比較的安全なファイルの使い方になる。

ポイント:

1. ファイルは、アプリディレクトリ内に作成する
2. ファイルのアクセス権は、他のアプリに対しては読み取り専用モードにする
3. センシティブな情報は格納しない
4. ファイルに格納する(された)情報に対しては、その入手先に関わらず内容の安全性を確認する

PublicFileActivity.java

```
package org.jssec.android.file.publicfile.readonly;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.publicfile.readonly.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicFileActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "public_file.dat";

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

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * ファイルの作成処理
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント 1★ ファイルは、アプリディレクトリ内に作成する
            // ★ポイント 2★ ファイルのアクセス権は、他のアプリに対しては読み取り専用モードにする
            fos = openFileOutput(FILE_NAME, MODE_WORLD_READABLE);

            // ★ポイント 3★ センシティブな情報は格納しない
            // ★ポイント 4★ ファイルに格納する情報に対しては、その入手先に関わらず内容の安全性を確認する
```

All rights reserved © Japan Smartphone Security Association.

```
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            fos.write(new String("センシティブでない情報(Public File Activity)¥n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
        }

        finish();
    }

    /**
     * ファイルの読み込み処理
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * ファイルの削除処理
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(this.getFilesDir() + "/" + FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
```

 All rights reserved © Japan Smartphone Security Association.

```
}
```

PublicUserActivity.java

```
package org.jssec.android.file.publicuser.readonly;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.publicuser.readonly.R;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicUserActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_PACKAGE = "org.jssec.android.file.publicfile.readonly";
    private static final String TARGET_CLASS = "org.jssec.android.file.publicfile.readonly.PublicFileActivity";

    private static final String FILE_NAME = "public_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        try {
            startActivity(intent);
        } catch (ActivityNotFoundException e) {
            mFileView.setText("(File Activity がありませんでした)");
        }
    }

    /**
     * ファイル Activity の呼び出し処理
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }
```

All rights reserved © Japan Smartphone Security Association.

```
    /**
     * ファイルの読み込み処理
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // ★ポイント 4★ ファイルに格納された情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * ファイルの追記処理
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        boolean exception = false;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            // 書き込みは失敗する。FileNotFoundException が発生
            fos = new FileOutputStream(file, true);

            fos.write(new String("センシティブでない情報(Public User Activity)¥n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(e.getMessage());
            exception = true;
        } catch (IOException e) {
            mFileView.setText(e.getMessage());
            exception = true;
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    exception = true;
                }
```

 All rights reserved © Japan Smartphone Security Association.

```
            }
        }

        if (exception == false)
            callFileActivity();
    }

    private final String getFilesPath(String filename) {
        String path = "";

        try {
            Context ctx = createPackageContext(TARGET_PACKAGE,
                    Context.CONTEXT_IGNORE_SECURITY);
            File file = new File(ctx.getFilesDir(), filename);
            path = file.getPath();
        } catch (NameNotFoundException e) {
        }
        return path;
    }
}
```

All rights reserved © Japan Smartphone Security Association.

4.6.1.3. 読み書き公開ファイルを扱う

不特定多数のアプリに対して、読み書き権限を許可するファイルの使い方である。

不特定多数のアプリが読み書き可能ということは、マルウェアも当然内容の書き換えが可能であり、データの信頼性も安全性も全く保証されない。また、悪意のない場合でもファイル内のデータの形式や書き込みを行うタイミングなど制御が困難であり、そのようなファイルは機能面からも実用性が無いに等しい。

以上のように、セキュリティの観点からもアプリ設計の観点からも、読み書き公開ファイルを安全に運用することは不可能であり、読み書き公開ファイルの使用は避けなければならない。

ポイント:

1. 他アプリから読み書き可能なアクセス権を設定したファイルは作らない

 All rights reserved © Japan Smartphone Security Association.

4.6.1.4. 外部記憶(読み書き公開)ファイルを扱う

SD カードのような外部記憶デバイス上にファイルを格納する場合である。比較的容量の大きな情報を格納する(Web からダウンロードしたファイルを置くなどの)場合や外部に情報を持ち出す(バックアップなどの)場合に利用することが想定される。

「外部記憶(読み書き公開)ファイル」は不特定多数のアプリに対して「読み取り公開ファイル」と同等の性質を持つ。さらに android.permission.WRITE_EXTERNAL_STORAGE Permission を利用宣言している不特定多数のアプリに対しては「読み書き公開ファイル」と同等の性質を持つ。そのため、外部記憶(読み書き公開)ファイルの使用は必要最小限にとどめるべきである。

Android アプリの慣例として、バックアップファイルは外部記憶デバイス上に作成されることが多い。しかし外部記憶デバイス上のファイルは前述のようにマルウェアを含む他のアプリから改ざんや削除されてしまうリスクがある。ゆえにバックアップを出力するアプリでは「バックアップファイルは速やかに PC 等の安全な場所にコピーしてください」といった警告表示をするなど、アプリの仕様や設計面でのリスク最小化の工夫も必要となる。

ポイント:
1. センシティブな情報は格納しない
2. アプリ毎にユニークなディレクトリにファイルを配置する
3. ファイルに格納する(された)情報に対しては、その入手先に関わらず内容の安全性を確認する
4. 利用側のアプリで書き込みを行わない仕様にする

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

    <uses-sdk android:minSdkVersion="8" />
    <!-- android.permission.WRITE_EXTERNAL_STORAGE Permission を利用宣言する -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ExternalFileActivity"
            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>
```

All rights reserved © Japan Smartphone Security Association.

ExternalFileActivity.java
```
package org.jssec.android.file.externalfile;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.externalfile.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ExternalFileActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_TYPE = "external";

    private static final String FILE_NAME = "external_file.dat";

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

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * ファイルの作成処理
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント 1★ センシティブな情報は格納しない
            // ★ポイント 2★ アプリ毎にユニークなディレクトリにファイルを配置する
            File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
            fos = new FileOutputStream(file, false);

            // ★ポイント 3★ ファイルに格納する情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            fos.write(new String("センシティブでない情報(External File Activity)¥n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
        }
```

 All rights reserved © Japan Smartphone Security Association.

```
        finish();
    }

    /**
     * ファイルの読み込み処理
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // ★ポイント 3★ ファイルに格納された情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * ファイルの削除処理
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
}
```

ExternalFileUser.java

```
package org.jssec.android.file.externaluser;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
```

All rights reserved © Japan Smartphone Security Association.

```
import org.jssec.android.file.externaluser.R;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ExternalUserActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_PACKAGE = "org.jssec.android.file.externalfile";
    private static final String TARGET_CLASS = "org.jssec.android.file.externalfile.ExternalFileActivity";
    private static final String TARGET_TYPE = "external";

    private static final String FILE_NAME = "external_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        try {
            startActivity(intent);
        } catch (ActivityNotFoundException e) {
            mFileView.setText("(File Activity がありませんでした)");
        }
    }

    /**
     * ファイル Activity の呼び出し処理
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * ファイルの読み込み処理
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getFilesPath(FILE_NAME));
```

 All rights reserved © Japan Smartphone Security Association.

```
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // ★ポイント 3★ ファイルに格納された情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * ファイルの追記処理
     *
     * @param view
     */
    public void onWriteFileClick(View view) {

        // ★ポイント 4★ 利用側のアプリで書き込みを行わない仕様にする
        // ただし、悪意のあるアプリが上書き・削除などを行うことを想定してアプリの設計を行うこと

        final AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(
                this);
        alertDialogBuilder.setTitle("ポイント 4");
        alertDialogBuilder.setMessage("利用側のアプリで書き込みを行わないこと");
        alertDialogBuilder.setPositiveButton("OK",
                new DialogInterface.OnClickListener() {

                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        callFileActivity();
                    }
                });

        alertDialogBuilder.create().show();

    }

    private final String getFilesPath(String filename) {
        String path = "";

        try {
            Context ctx = createPackageContext(TARGET_PACKAGE,
                    Context.CONTEXT_IGNORE_SECURITY);
            File file = new File(ctx.getExternalFilesDir(TARGET_TYPE), filename);
            path = file.getPath();
        } catch (NameNotFoundException e) {
```

All rights reserved © Japan Smartphone Security Association.

```
        }
        return path;
    }
}
```

 All rights reserved © Japan Smartphone Security Association.

4.6.2. ルールブック

ファイルを扱う場合には以下のルールを守ること。

1.	ファイルは原則非公開ファイルとして作成する	(必須)
2.	他のアプリから読み書き権限でアクセス可能なファイルは作成しない	(必須)
3.	SD カードなど外部記憶デバイスに格納するファイルの利用は必要最小限にする	(必須)
4.	ファイルの生存期間を考慮してアプリの設計を行う	(必須)

4.6.2.1. ファイルは原則非公開ファイルとして作成する (必須)

「4.6 ファイルを扱う」「4.6.1.1 非公開ファイルを扱う」で述べたように、格納する情報の内容に関わらずファイルは原則非公開にするべきである。Android のセキュリティ設計の観点からも、情報のやり取りとそのアクセス制御は Content Provider や Service などの Android の仕組みの中で行うべきであり、できない事情がある場合のみファイルのアクセス権で代用することを検討することになる。

各ファイルタイプのサンプルコードや以下のルールの項も参照のこと。

4.6.2.2. 他のアプリから読み書き権限でアクセス可能なファイルは作成しない (必須)

「4.6.1.3 読み書き公開ファイルを扱う」で述べたように、他のアプリに対してファイルの読み書きを許可すると、ファイルに格納される情報の制御ができない。そのため、セキュリティ的な観点からも機能・設計的な観点からも読み書き公開ファイルを利用した情報の共有を考えるべきではない。

4.6.2.3. SD カードなど外部記憶デバイスに格納するファイルの利用は必要最小限にする (必須)

「4.6.1.4 外部記憶(読み書き公開)ファイルを扱う」で述べたように、SD カードをはじめとする外部記憶デバイスにファイルを置くことは、セキュリティおよび機能の両方の観点から潜在的な問題を抱えることに繋がる。一方で、SD カードはアプリディレクトリより生存期間の長いファイルを扱え、アプリ外部にデータを持ち出すのに常時使える唯一のストレージなので、アプリの仕様によっては使用せざる得ないケースも多いと考えられる。

外部記憶デバイスにファイルを格納する場合、不特定多数のアプリおよびユーザーが読み・書き・削除できることを考慮して、サンプルコードで述べたポイントを含めて以下のようなポイントに気をつけてアプリの設計を行う必要がある。なお暗号化や電子署名などの暗号技術については本ガイドの今後の版で別途記事を設ける予定である。

- 原則としてセンシティブな情報は外部記憶デバイス上のファイルに保存しない
- もしセンシティブな情報を外部記憶デバイス上のファイルに保存する場合は暗号化する
- 他アプリやユーザーに改ざんされては困る情報を外部記憶デバイス上のファイルに保存する場合は電子署名も一緒に保存する
- 外部記憶デバイス上のファイルを読み込む場合、読み込むデータの安全性を確認してからデータを利用する
- 他のアプリやユーザーによって外部記憶デバイス上のファイルはいつでも削除されることを想定してアプリを設

All rights reserved © Japan Smartphone Security Association.

計しなければならない

「4.6.2.4 ファイルの生存期間を考慮してアプリの設計を行う　（必須）」も参照すること。

4.6.2.4. ファイルの生存期間を考慮してアプリの設計を行う　（必須）

アプリディレクトリに保存されたデータは以下のユーザー操作により消去される。アプリの生存期間と一致する、またはアプリの生存期間より短いのが特徴である。

- アプリのアンインストール
- 各アプリのデータおよびキャッシュの消去（「設定」→「アプリケーション」→「アプリケーションの管理」）

SD カード等の外部記憶デバイス上に保存されたファイルは、アプリの生存期間よりファイルの生存期間が長いことが特徴である。さらに次の状況も想定する必要がある。

- ユーザーによるファイルの消去
- SD カードの抜き取り・差し替え・アンマウント
- マルウェアによるファイルの消去

このようにファイルの保存場所によってファイルの生存期間が異なるため、本節で説明したようなセンシティブな情報を保護する観点だけでなく、アプリとして正しい動作を実現する観点でもファイルの保存場所を正しく選択する必要がある。

All rights reserved © Japan Smartphone Security Association.

4.6.3. アドバンスト

4.6.3.1. ファイルディスクリプタ経由のファイル共有

他のアプリに公開ファイルを直接アクセスさせるのではなく、ファイルディスクリプタ経由でファイル共有する方法がある。Content Provider と Service でこの方法が使える。Content Provider や Service の中で非公開ファイルをオープンし、そのファイルディスクリプタを相手のアプリに渡す。相手アプリはファイルディスクリプタ経由でファイルを読み書きできる。

他のアプリにファイルを直接アクセスさせるファイル共有方法とファイルディスクリプタ経由のファイル共有方法の比較を表 4.6.3-1 に示す。アクセス権のバリエーションとアクセス許可するアプリの範囲でメリットがある。特にアクセスを許可するアプリを細かく制御できるところがセキュリティ観点ではメリットが大きい。

表 4.6.3-1 アプリ間ファイル共有方法の比較

ファイル共有方法	アクセス権設定のバリエーション	アクセスを許可するアプリの範囲
他のアプリにファイルを直接アクセスさせるファイル共有	読み取り 書き込み 読み取り＋書き込み	すべてのアプリに対して一律アクセス許可してしまう
ファイルディスクリプタ経由のファイル共有	読み取り 書き込み 追記のみ 読み取り＋書き込み 読み取り＋追記のみ	Content Provider や Service にアクセスしてくるアプリに対して個別に一時的にアクセス許可・不許可を制御できる

上記ファイル共有方法のどちらにも共通することであるが、他のアプリにファイルの書き込みを許可するとファイル内容の完全性が保証しづらくなる。特に複数のアプリから同時に書き込みが行われると、ファイル内容のデータ構造が壊れてしまいアプリが正常に動作しなくなるリスクがある。他のアプリとのファイル共有においては、読み込み権限だけを許可するのが望ましい。

以下では、Content Provider でファイルを共有する実装例(非公開 Provider の場合)をサンプルコードとして掲載する。

ポイント

1. 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保存してよい
2. 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

ProprietaryProvider.java
```
package org.jssec.android.file.proprietaryprovider;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
```

All rights reserved © Japan Smartphone Security Association.

```
import java.io.IOException;

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

import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.os.ParcelFileDescriptor;

public class ProprietaryProvider extends ContentProvider {

    private static final String FILENAME = "sensitive.txt";

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.file.proprietaryprovider.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() {
        File dir = getContext().getFilesDir();
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(new File(dir, FILENAME));
            // ★ポイント 1★ 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保存してよい
            fos.write(new String("センシティブな情報").getBytes());

        } catch (IOException e) {
        } finally {
            try {
                fos.close();
            } catch (IOException e) {
            }
        }

        return true;
    }

    @Override
    public ParcelFileDescriptor openFile(Uri uri, String mode)
            throws FileNotFoundException {

        // 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
```

 All rights reserved © Japan Smartphone Security Association.

```
        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。");
        }

        File dir = getContext().getFilesDir();
        File file = new File(dir, FILENAME);

        // サンプルのため読み取り専用を常に返す
        int modeBits = ParcelFileDescriptor.MODE_READ_ONLY;
        return ParcelFileDescriptor.open(file, modeBits);
    }

    @Override
    public String getType(Uri uri) {
        return "";
    }

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

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
        return 0;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }
}
```

ProprietaryUserActivity.java

```
package org.jssec.android.file.proprietaryprovideruser;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import org.jssec.android.file.proprietaryprovideruser.R;
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.pm.PackageManager;
```

All rights reserved © Japan Smartphone Security Association.

```
import android.content.pm.ProviderInfo;
import android.net.Uri;
import android.os.Bundle;
import android.os.ParcelFileDescriptor;
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 AUTHORITY = "org.jssec.android.file.proprietaryprovider";

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.file.proprietaryprovider.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, String authority) {
        String pkgname = null;
        PackageManager pm = context.getPackageManager();
        ProviderInfo pi = pm.resolveContentProvider(authority, 0);
        if (pi != null)
            pkgname = pi.packageName;
        return pkgname;
    }

    public void onReadFileClick(View view) {

        logLine("[ReadFile]");

        // 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;
        }

        // 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, AUTHORITY);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // 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
        ParcelFileDescriptor pfd = null;
```

 All rights reserved © Japan Smartphone Security Association.

```
        try {
            pfd = getContentResolver().openFileDescriptor(
                    Uri.parse("content://" + AUTHORITY), "r");
        } catch (FileNotFoundException e) {

        }

        if (pfd != null) {
            FileInputStream fis = new FileInputStream(pfd.getFileDescriptor());

            if (fis != null) {
                try {
                    byte[] buf = new byte[(int) fis.getChannel().size()];
                    fis.read(buf);
                    // ★ポイント 2★ 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。「3.2 入力データの安全性を確認する」を参照。
                    logLine(new String(buf));
                } catch (IOException e) {
                } finally {
                    try {
                        fis.close();
                    } catch (IOException e) {
                    }
                }
            }
            try {
                pfd.close();
            } catch (IOException e) {
            }

        } else {
            logLine("  null file descriptor");
        }
    }

    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");
    }
}
```

4.6.3.2. ディレクトリのアクセス権設定

これまでファイルに着目してセキュリティの考慮点を説明してきた。ファイルのコンテナであるディレクトリについてもセキュリティの考慮が必要である。ここではディレクトリのアクセス権設定についてセキュリティ上の考慮ポイントを説明する。

Android には、アプリディレクトリ内にサブディレクトリを取得・作成するメソッドがいくつか用意されている。主なものを表 4.6.3-2 に示す。

表 4.6.3-2 アプリディレクトリ配下のサブディレクトリ取得・作成メソッド

	他アプリに対するアクセス権の指定	ユーザーによる削除
Context#getFilesDir()	不可(実行権限のみ)	「設定」→「アプリケーション」→「アプリケーションの管理」→「データ消去」
Context#getCacheDir()	不可(実行権限のみ)	「設定」→「アプリケーション」→「アプリケーションの管理」→「キャッシュ消去」 ※「データ消去」でも削除される。
Context#getDir(String name, int mode)	mode に以下を指定可能。 MODE_PRIVATE MODE_WORLD_READABLE MODE_WORLD_WRITABLE	「設定」→「アプリケーション」→「アプリケーションの管理」→「データ消去」

ここで特に気を付けるのは Context#getDir()によるアクセス権の設定である。ファイルの作成でも説明しているように、Android のセキュリティ設計の観点からディレクトリも基本的には非公開にするべきであり、アクセス権の設定によって情報の共有を行うと思わぬ副作用があるので、情報の共有には他の手段を考えるべきである。

MODE_WORLD_READABLE

すべてのアプリに対してディレクトリの読み取り権限を与えるフラグである。すべてのアプリがディレクトリ内のファイル一覧や個々のファイルの属性情報を取得可能になる。このディレクトリ配下に秘密のファイルを配置することはできないので、このフラグの使用には十分な注意が必要である。

MODE_WORLD_WRITABLE

他アプリに対してディレクトリの書き込み権限を与えるフラグである。すべてのアプリがディレクトリ内のファイルを作成、移動[8]、リネーム、削除が可能になる。これらの操作はファイル自体のアクセス権設定（読み取り、書き込み、実行）とは無関係であり、ディレクトリの書き込み権限があるだけで可能となる操作であることに注意が必要だ。他のアプリから勝手にファイルを削除されたり置き換えられたりするので、通常はこのフラグを使用してはならない。

表 4.6.3-2 の「ユーザーによる削除」に関しては、「4.6.2.4 ファイルの生存期間を考慮してアプリの設計を行う（ 必須）」を参照のこと。

4.6.3.3. Shared Preference やデータベースファイルのアクセス権設定

[8] 内部ストレージから外部記憶装置(SD カードなど)への移動などマウントポイントを超えた移動はできない。そのため、読み取り権限のない内部ストレージファイルが外部記憶装置に移動されて読み書き可能になるようなことはない。

 All rights reserved © Japan Smartphone Security Association.

Shared Preference やデータベースもファイルで構成される。アクセス権設定についてはファイルと同じことが言える。したがって Shared Preference もデータベースもファイルと同様に基本的には非公開ファイルとして作成し、内容の共有は Android のアプリ間連携の仕組みによって実現するべきである。

Shared Preference の使用例を次に示す。MODE_PRIVATE により非公開ファイルとして Shared Preference を作成している。

Shared Preference ファイルにアクセス制限を設定する例

```
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;

【中略】

        // Shared Preference を取得する（なければ作成される）
        // ポイント：基本的に MODE_PRIVATE モードを指定する
        SharedPreferences preference = getSharedPreferences(
                PREFERENCE_FILE_NAME, MODE_PRIVATE);

        // 値が文字列のプリファレンスを書き込む例
        Editor editor = preference.edit();
        editor.putString("prep_key", "prep_value");// key:"prep_key", value:"prep_value"
        editor.commit();
```

データベースについては「4.5 SQLite を使う」を参照すること。

All rights reserved © Japan Smartphone Security Association.

4.7. Browsable Intent を利用する

ブラウザから Web ページのリンクに対応して起動するようにアプリを作ることができる。Browsable Intent という機能である。アプリは、URI スキームを Manifest ファイルで指定することで、その URI スキームを持つリンクへの移動(ユーザーのタップなど)に反応し、リンクをパラメータとして起動することが可能になる。

また、URI スキームを利用することでブラウザから対応するアプリを起動する方法は、Android のみならず iOS 他のプラットフォームでも対応しており、Web アプリとの外部アプリ連携などに一般的に使われている。例えば、Twitter アプリや Facebook アプリでは次のような URI スキームが定義されており、Android でも iOS でもブラウザから対応するアプリが起動するようになっている。

URI スキーム	対応するアプリ
fb://	Facebook
twitter://	Twitter

このように連携や利便性を考えた便利な機能であるが、悪意ある第三者に悪用される危険性も潜んでいる。
悪意のある Web サイトを用意してリンクの URL に不正なパラメータを仕込むことでアプリの機能を悪用したり、同じ URI スキームに対応したマルウェアをインストールさせて URL に含まれる情報を横取りするなどが考えられる。

このような危険性に対応するために、利用する際にはいくつかのポイントに気をつけなければならない。

4.7.1. サンプルコード

以下に、Browsable Intent を利用したアプリのサンプルコードを示す。

ポイント:
1. (Web ページ側)対応する URI スキーマを使ったリンクのパラメータにセンシティブな情報を含めない
2. URL のパラメータを利用する前に値の安全性を確認する

Starter.html
```
<html>
    <body>
<!-- ★ポイント 1★ URL にセンシティブな情報を含めない -->
<!-- URL パラメータとして渡す文字列は、UTF-8 で、かつ URI エンコードしておくこと -->
        <a href="secure://jssec?user=user_id"> Login </a>
    </body>
</html>
```

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

 All rights reserved © Japan Smartphone Security Association.

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

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

            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                // 暗黙的 intent を受け付ける
                <category android:name="android.intent.category.DEFAULT" />
                // Browsable intent を受け付ける
                <category android:name="android.intent.category.BROWSABLE" />
                // URI 'secure://jssec' を受け付ける
                <data android:scheme="secure" android:host="jssec"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

BrowsableIntentActivity.java

```
package org.jssec.android.browsableintent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

public class BrowsableIntentActivity extends Activity {

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

        Intent intent = getIntent();
        Uri uri = intent.getData();
        if (uri != null) {
            // URL パラメータで渡されたユーザーID を取得する
            // ★ポイント 2★ URL のパラメータを利用する前に値の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String userID = "User ID = " +  uri.getQueryParameter("user");
            TextView tv = (TextView)findViewById(R.id.text_userid);
            tv.setText(userID);
        }
    }

}
```

All rights reserved © Japan Smartphone Security Association.

4.7.2. ルールブック

Browsable Intent を利用する場合には以下のルールを守ること。

1.	(Web ページ側)対応するリンクのパラメータにセンシティブな情報を含めない	（必須）
2.	URL のパラメータを利用する前に値の安全性を確認する	（必須）

4.7.2.1. (Web ページ側)対応するリンクのパラメータにセンシティブな情報を含めない （必須）

ブラウザ上でリンクをタップした際、data（Intent#getData にて取得)に URL の値が入った Intent が発行され、システムにより該当する Intent Filter を持つアプリが起動する。
この時、同じ URI スキームを受け付けるよう Intent Filter が設定されたアプリが複数存在する場合は、通常の暗黙的 Intent による起動と同様にアプリ選択のダイアログが表示され、ユーザーの選択したアプリが起動することになる。
仮に、アプリ選択画面の選択肢としてマルウェアが存在していた場合は、ユーザーが誤ってマルウェアを起動させてしまう危険性があり、パラメータがマルウェアに渡ることになる。

このように Web ページのリンク URL に含めたパラメータはすべてマルウェアに渡る可能性があるので、一般の Web ページのリンクを作るときと同様に、URL のパラメータに直接センシティブな情報を含めることは避けなければならない。

URL にユーザーID とパスワードが入っている例
```
insecure://sample/login?userID=12345&password=abcdef
```

また、URL のパラメータがユーザーID などセンシティブでない情報のみの場合でも、アプリ起動時のパスワード入力をアプリ側でさせるような仕様では、ユーザーが気付かずにマルウェアを起動してしまい、マルウェアに対してパスワードを入力してしまう危険性もある。そのため、一連のログイン処理自体はアプリ側で完結するような仕様を検討すべきである。Browsable Intent によるアプリ起動はあくまで暗黙的 Intent によるアプリ起動であり、意図したアプリが起動される保証がないことを念頭に置いたアプリ・サービス設計を心がける必要がある。

4.7.2.2. URL のパラメータを利用する前に値の安全性を確認する （必須）

URI スキーマに合わせたリンクは、アプリ開発者に限らず誰でも作成可能なので、アプリに渡された URL のパラメータが正規の Web ページから送られてくるとは限らない。また、渡された URL のパラメータが正規の Web ページから送られてきたかどうかを調べる方法もない。

そのため、渡された URL のパラメータを利用する前に、パラメータに想定しない値が入っていないかなど、値の安全性を確認する必要がある。

 All rights reserved © Japan Smartphone Security Association.

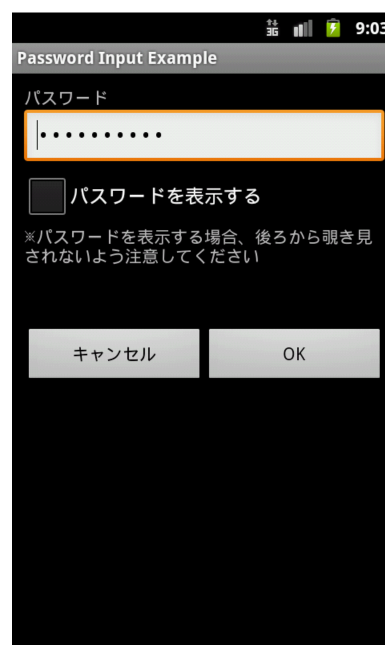
5. セキュリティ機能の使い方

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今後の版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

All rights reserved © Japan Smartphone Security Association.

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 つのメソッド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());
```

All rights reserved © Japan Smartphone Security Association.

```
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        // 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);

        // 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) {
```


All rights reserved © Japan Smartphone Security Association.

```
                // 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);
        }

    }

    // 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
```

All rights reserved © Japan Smartphone Security Association.

```
        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。アプリの仕様として、パスワードをマスク表示する機能が必要である。

パスワードを入力する EditText をマスク表示する方法には、静的にレイアウト XML で指定する方法と、動的にプログラム上で切り替える方法の 2 種類がある。前者は、レイアウト XML の EditText タグで android:password 属性を android:password="true"にすることで実現できる。後者は、EditText クラスの setInputType()メソッドで EditText の入力タイプに InputType.TYPE_TEXT_VARIATION_PASSWORD を追加することで実装できる。

以下、それぞれのサンプルコードを示す。

レイアウト XML で指定する方法

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 内で指定する方法

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);
```

All rights reserved © Japan Smartphone Security Association.

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;
```

 All rights reserved © Japan Smartphone Security Association.

```
            } 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
```

All rights reserved © Japan Smartphone Security Association.

```
     *
     * @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;
```


All rights reserved © Japan Smartphone Security Association.

```
        }

        ～ 省略 ～

    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        // 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);

        // 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
        }

    }
```

All rights reserved © Japan Smartphone Security Association.

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か、十分なセキュリティ対策が可能な場合にのみ、自動ログイン機能は利用することができる。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に合わせてセキュリティ対策が必須となる。対策としては、「決済処理などの金銭的な処理が発生する直前にはパスワードの再入力を求める」、「自動ログイン設定時にユーザーに対して十分に注意を喚起し、確実な端末のロックを促す」などいくつか考えられる。自動ログインの利用にあたっては、これらの対策を前提に利便性とリスクを勘案して、慎重な検討を行うべきである。

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。

All rights reserved © Japan Smartphone Security Association.

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。

All rights reserved © Japan Smartphone Security Association.

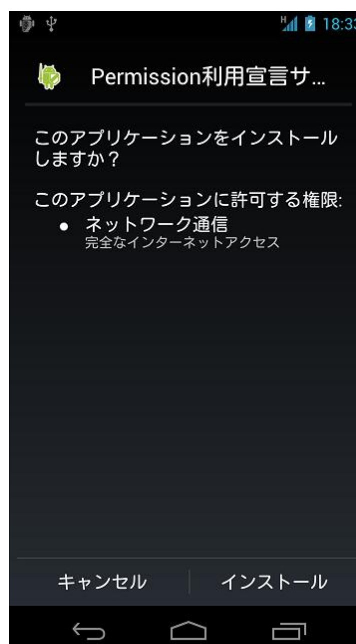
5.2. Permission と Protection Level

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"?>
```

 All rights reserved © Japan Smartphone Security Association.

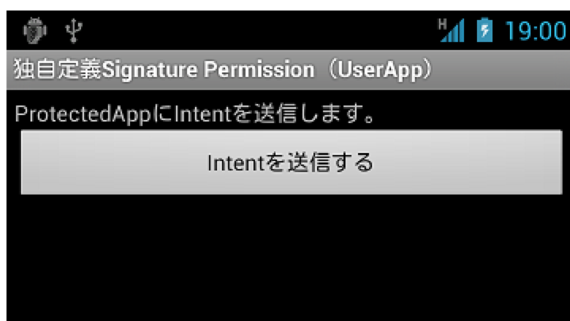
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.usespermission"
    android:versionCode="1"
    android:versionName="1.0" >

    <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>
```

All rights reserved © Japan Smartphone Security Association.

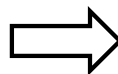
5.2.1.2. 独自定義の Signature Permission で自社アプリ連携する方法

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.protectedapp.MY_PERMISSION"
        android:protectionLevel="signature" />

    <application
```

 All rights reserved © Japan Smartphone Security Association.

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

        <!-- ★ポイント 2★ Component には 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.protectedapp.MY_PERMISSION" >

            <!-- ★ポイント 3★ Component が Activity の場合には 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.protectedapp.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★ 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
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            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

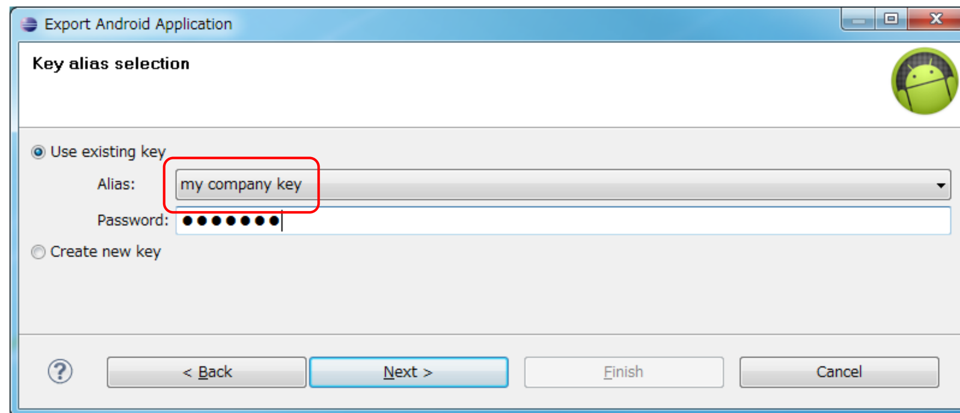
```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
```

 All rights reserved © Japan Smartphone Security Association.

```
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

All rights reserved © Japan Smartphone Security Association.

★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

6. 独自定義 Signature Permission は定義しない
7. uses-permission により独自 Permission を利用宣言する
8. 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
9. 利用先アプリが自社アプリであることを確認する
10. 利用先 Component が Activity の場合、明示的 Intent を使う
11. 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 6★ 独自定義 Signature Permission は定義しない -->

    <!-- ★ポイント 7★ uses-permission により独自 Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.permission.protectedapp.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>
```

 All rights reserved © Japan Smartphone Security Association.

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.protectedapp.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 8★ 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
        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 9★ 利用先アプリが自社アプリ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;
        }
```

All rights reserved © Japan Smartphone Security Association.

```
        // ★ポイント 10★ 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

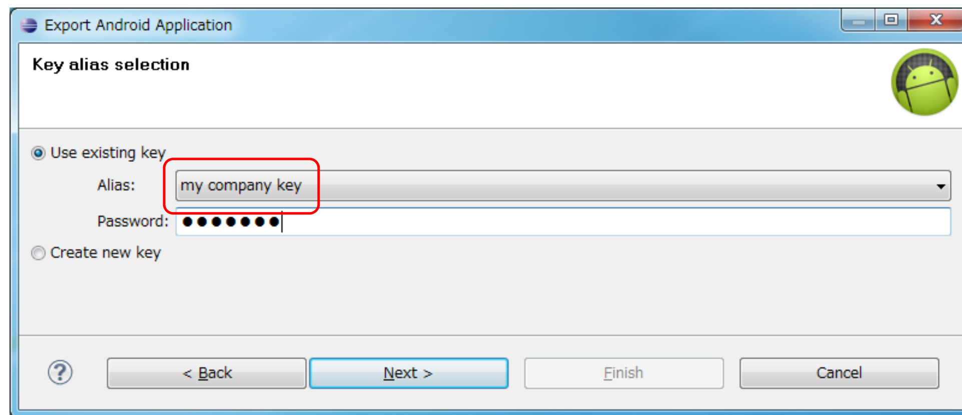
```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
```

 All rights reserved © Japan Smartphone Security Association.

```
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

All rights reserved © Japan Smartphone Security Association.

★ポイント 11★ 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。



All rights reserved © Japan Smartphone Security Association.

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。

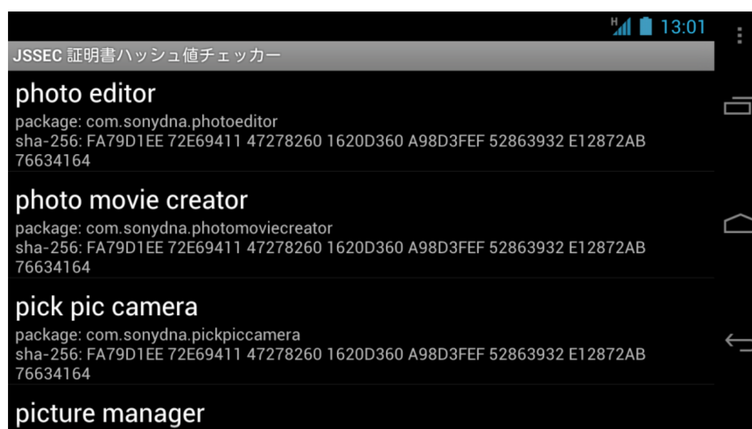
Android のデバッグ証明書の内容を keytool で出力する例

```
> 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


*******************************************
*******************************************
```

All rights reserved © Japan Smartphone Security Association.

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。

 All rights reserved © Japan Smartphone Security Association.

5.2.2. ルールブック

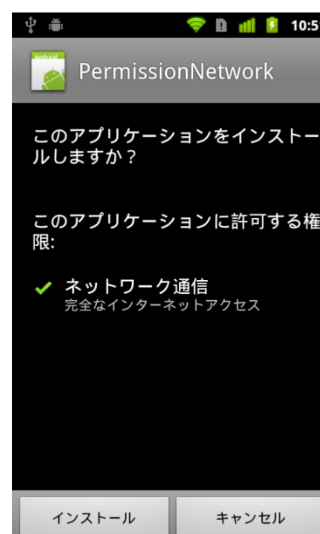
独自 Permission 利用時には以下のルールを守ること。

1.	Android OS 規定の Dangerous Permission はユーザーの資産を保護するためにだけ利用する	（必須）
2.	独自定義の Dangerous Permission は利用してはならない	（必須）
3.	独自定義 Signature Permission は Component の提供側アプリでのみ定義する	（必須）
4.	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）
5.	独自定義の Normal Permission は利用してはならない	（推奨）
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. Android OS 規定の Dangerous Permission はユーザーの資産を保護するためにだけ利用する（必須）

独自定義の Dangerous Permission の利用は非推奨（「5.2.2.2 独自定義の Dangerous Permission は利用してはならない　（必須）」参照）のため、ここでは Android OS 規定の Dangerous Permission を前提に話をする。

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。



アプリの中には、ユーザーの資産とアプリ開発者が保護したい資産がある。それらのうち、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。なぜなら、権限の付与がユーザーの判断に委ねられているためである。一方、アプリ開発者が保護したい資産については、この方法では保護できない。

例えば、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場合を考える。このような Component を Dangerous Permission により保護するように実装したとする。他社アプリがインストールされると

All rights reserved © Japan Smartphone Security Association.

きに、ユーザーの判断により他社アプリに対して権限の付与を許可してしまうと、保護すべき自社の資産が他社アプリに悪用される危険が生じる。このような場合に自社の資産を保護するためには、独自定義の Signature Permission を使うとよい。

5.2.2.2. 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 を使用しても、インストール時に「ユーザーに権限の許可を求める」画面が表示されない場合がある。つまり Dangerous Permission の特徴であるユーザーに判断を求める機能が働かないことがあるのだ。よって本ガイドでは「独自定義の Dangerous Permission を利用しない」ことをルールとする。

まず説明のために 2 つのアプリを想定する。1 つは独自の Dangerous Permission を定義し、この Permission により保護した Component を公開するアプリである。これを ProtectedApp とする。もう 1 つは ProtectedApp の Component を悪用しようとする別のアプリでこれを AttackerApp とする。ここで AttackerApp は ProtectedApp が定義した Permission の利用宣言とともに同じ Permission の定義も行っているものとする。

AttackerApp がユーザーの許可なしに ProtectedApp の Component を利用できてしまうケースは以下のような場合に起きる。

1. ユーザーがまず AttackerApp 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が完了してしまう
2. 次に ProtectedApp をインストールすると、ここでも特に警告もなくインストールできてしまう
3. その後、ユーザーが AttackerApp を起動すると、AttackerApp はユーザーの気づかぬうちに ProtectedApp の Component にアクセスできてしまい、場合によっては被害に繋がる

このケースの原因は次の通りである。先に AttackerApp 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、このケースで Component へのアクセスを防止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。

5.2.2.3. 独自定義 Signature Permission は Component の提供側アプリでのみ定義する （必須）

 All rights reserved © Japan Smartphone Security Association.

自社アプリ間で連携する場合、実行時にSignature Permissionをチェックすることでセキュリティを担保できることを「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」で例示した。この仕組みを利用する際には、Protection Level が Signature の独自 Permission の定義は、Component 提供側アプリの AndroidManifest.xml でのみ行い、利用側アプリでは独自の Signature Permission を定義してはならない。

以下がその理由となる。

提供側アプリより先にインストールされた利用側アプリが複数あり、どの利用側アプリも独自定義 Permission の利用宣言とともに Permission の定義もしている場合を考える。この状況で提供側アプリをインストールすると、すべての利用側アプリから提供側アプリにアクセスすることが可能になる。次に、最初にインストールした利用側アプリをアンインストールすると、Permission の定義が削除され、Permission が未定義となる。そのため、残った利用側アプリからの提供側アプリの利用が不可能となってしまう。

このように、利用側アプリで Permission の定義を行うと思わぬ Permission の未定義状態が発生するので、Permission の定義は保護する Component の提供側アプリのみ行い、利用側アプリで Permission を定義するのは避けなければならない。

こうすることで、提供側アプリのインストール時に権限付与が行われ、かつ、アンインストール時に Permission が未定義となり、提供側アプリと Permission の定義の存在期間が必ず一致するので、適正な Component の提供と保護が可能である。 なお、独自定義 Signature Permission に関しては、連携するアプリのインストール順によらず、利用側アプリに Permission 利用権限が付与されるため(Android 2.2 以降)、この議論が成り立つことに注意[9]。

5.2.2.4. 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、Component 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「5.2.3.1 独自定義 Signature Permission を回避できる Android OS の特性とその対策」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、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）

[9] Signature Permission による保護を Android 2.1 以前(本ガイドのスコープ外)の端末で利用する場合、あるいは、Normal/Dangerous Permission を利用する場合には、Permission が未定義のまま利用側アプリが先にインストールされると、利用側アプリへの権限の付与が行われず、提供側アプリがインストールされた後もアクセスができない

All rights reserved © Japan Smartphone Security Association.

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” />

続いて、ソースコード上にて次を実装する。

4. 保護したい Component でリクエストを処理する前に、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 にアクセスする前に、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。

5.2.2.5. 独自定義の Normal Permission は利用してはならない （推奨）

Normal Permission を利用するアプリは Android Manifest.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 Normal Permission は利用できない。

さらに独自定義 Normal Permission を用いてアプリ間連携を行う場合、連携する各アプリへの Permission の付与はインストール順に依存する。例えば、Permission を定義したコンポーネントを持つアプリよりも先にその Permission を利用宣言したアプリをインストールすると、Permission を定義したアプリをインストールした後も利用側アプリは Permission で保護されたコンポーネントにアクセスすることができない。

インストール順によりアプリ間連携ができなくなる問題を回避する方法として、連携する全てのアプリに Permission を定義することも考えられる。そうすることにより最初に利用側アプリがインストールされた場合でも、全ての利用側アプリが提供側アプリにアクセスすることが可能となる。しかし、最初にインストールした利用側アプリがアンインストールされた際に Permission が未定義な状態となり、他に利用側アプリが存在していても、それらのアプリから提供側アプリにアクセスすることができなくなってしまうのである。

以上のようにアプリの可用性が損なわれる恐れがあることから、独自定義 Normal Permission の利用は控えるべきである。

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）

 All rights reserved © Japan Smartphone Security Association.

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
```

All rights reserved © Japan Smartphone Security Association.

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 の特性とその対策

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。

まずは、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 の基本的な使い方を紹介する。ただし、後述するように、この使い方には Permission 回避の問題があることが分かっており、本ガイドに掲載した対策が必要となる。

以下、独自定義 Signature Permission の基本的な使い方である。

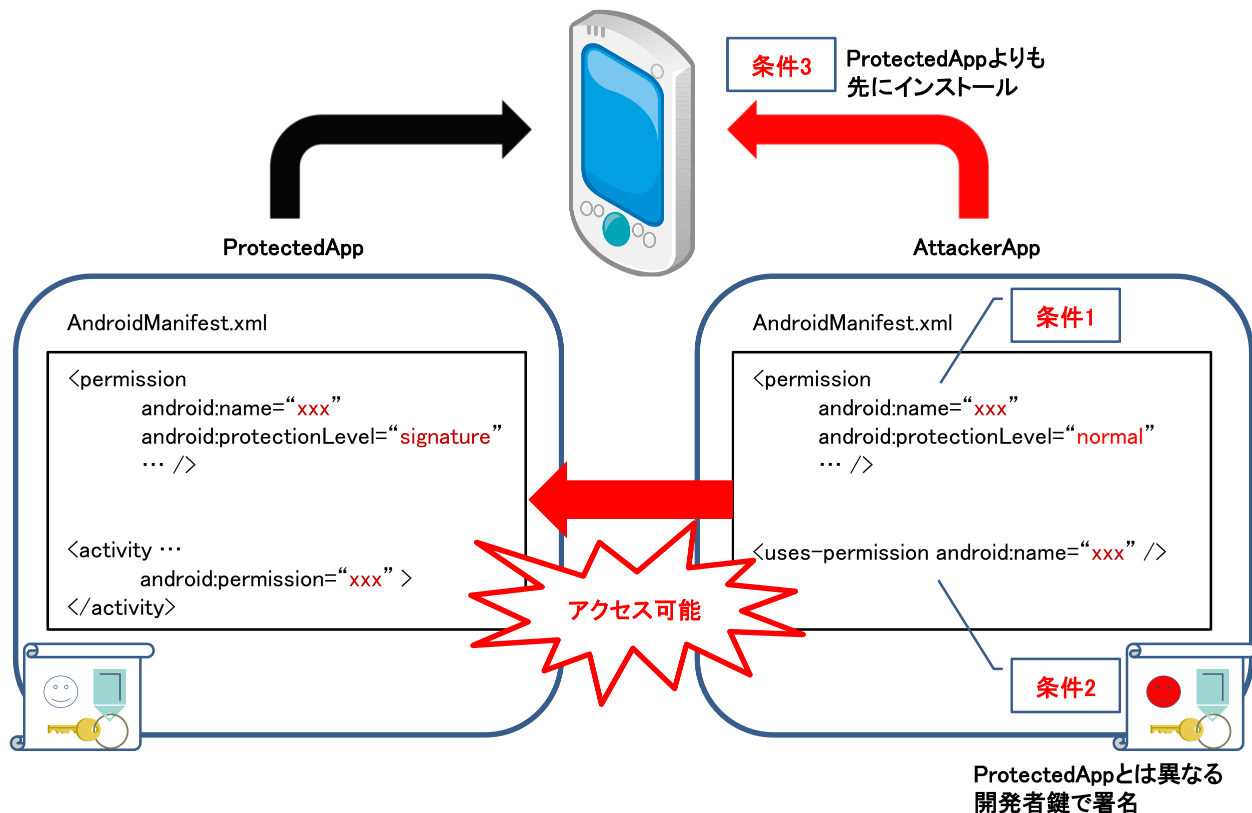
1. 連携するすべてのアプリの AndroidManifest.xml で、独自 Signature Permission を定義する
   例：<permission android:name=”xxx” android:protectionLevel=”signature” />
2. 保護したい Component を持つアプリの AndroidManifest.xml で、保護したい Component に android:permission 属性を指定し、1.で定義した Signature Permission を要求する
   例：<activity android:permission=”xxx” ... >...</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、この使い方だけでは、次の条件が成立すると Signature Permission 回避の抜け道ができてしまう。

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。ここで Signature Permission 回避の抜け道とは、AttackerApp は署名が一致していないにもかかわらず、ProtectedApp の Component にアクセス可能になることである。

条件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）
   例: <permission android:name=" xxx " android:protectionLevel="normal" />
条件2. AttackerApp は独自定義した Normal Permission を uses-permission で利用宣言する
   例: <uses-permission android:name="xxx" />
条件3. AttackerApp を ProtectedApp より先に端末にインストールする

 All rights reserved © Japan Smartphone Security Association.



条件 1 および条件 2 の成立に必要な ProtectedApp 独自定義の Permission 名は、APK ファイルから AndroidManifest.xml を取り出せば攻撃者にとって容易に知ることができる。条件 3 もユーザーを騙すなどの方法により攻撃者にある程度制御の余地がある。

このように独自定義の Signature Permission には基本的な使い方だけでは保護を回避されてしまう危険性があり、抜け道をふさぐような対策が必要である。具体的にはルールセクションの「5.2.2.4 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)」に掲載している方法で対処できるので、そちらを参照のこと。

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、署名するアプリが配布されているため

All rights reserved © Japan Smartphone Security Association.

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();

        // 改ざんを検知する
        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
```

 All rights reserved © Japan Smartphone Security Association.

```
                        return false;
                    }
                }
            }

            if(permissionList.size() == 0){
                // OK
                return true;
            }

        } catch (NameNotFoundException e) {
        }

        return false;
    }
}
```

5.2.3.3. APK の改ざんを検出する

「5.2.3.2 ユーザーが AndroidManifest.xml を改ざんする」ではユーザーによる Permission 改ざんの検出について説明した。しかし、アプリの改ざんは Permission に限らず、リソースを差し替えて別のアプリとしてマーケットで配布するなど、ソースコードを変更することなく改ざんし流用する事例が多様に存在する。ここでは APK ファイルが改ざんされたことを検出するためのより汎用的な方法を紹介する。

APK の改ざんを行うには、APK ファイルを一度展開し、内容を改変した後に再び APK ファイルとして再構成する必要がある。その際に改ざん者は元の開発者の鍵を持ち得ないので、改ざん者自身の鍵で APK を署名することになる。このようにAPKの改ざんには署名(証明書)の変更を伴うため、アプリ起動時にAPKの証明書と予めソースコードに埋め込んだ開発者の証明書を比較することで改ざんの有無を検出することができる。

以下にサンプルコードを示す。なお、実装例のままではプロのハッカーであれば改ざん検出の無効化が容易である。あくまで簡易な実装例であることを念頭においてアプリへの適用を検討するべきである。

ポイント：

1. 主要な処理を行うまでの間に、アプリの証明書が開発者の証明書であることを確認する

SignatureCheckActivity.java

```
package org.jssec.android.permission.signcheckactivity;

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

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

public class SignatureCheckActivity extends Activity {
    // 自己証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
```

All rights reserved © Japan Smartphone Security Association.

```
        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 1★ 主要な処理を行うまでの間に、アプリの証明書が開発者の証明書であることを確認する
        if (!PkgCert.test(this, this.getPackageName(), myCertHash(this))) {
            Toast.makeText(this, "自己署名の照合　NG", Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        Toast.makeText(this, "自己署名の照合　OK", Toast.LENGTH_LONG).show();
    }
}
```

PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
```

 All rights reserved © Japan Smartphone Security Association.

```
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

5.2.3.4. Permission の再委譲問題

アプリが Android OS に保護されている電話帳や GPS といった情報や機能にアクセスするためには Permission を利用宣言しなければならない。Permission を利用宣言し許可されると、そのアプリにはその Permission が委譲されたことになり、その Permission により保護された情報や機能にアクセスできるようになる。

プログラムの組み方によっては、Permission を委譲された（許可された）アプリは Permission で保護されたデータを取得し、そのデータを別のアプリに何の Permission も要求せずに提供することもできてしまう。これは Permission を持たないアプリが Permission で保護されたデータにアクセスできることに他ならない。実質的に Permission を再委譲していることと等価になるので、これを Permission の再委譲問題と呼ぶ。このように Android の Permission セキュリティモデルでは、保護されたデータへのアプリからの直接アクセスだけしか権限管理ができないという仕様上の性質がある。

具体例を図 5.2-1 に示す。中央のアプリは android.permission.READ_CONTACTS を利用宣言したアプリが連絡先情報を読み取って自分の DB に蓄積している。何の制限もなく Content Provider 経由で蓄積した情報を他のアプリに提供した場合に、Permission の再委譲問題が生じる。

All rights reserved © Japan Smartphone Security Association.

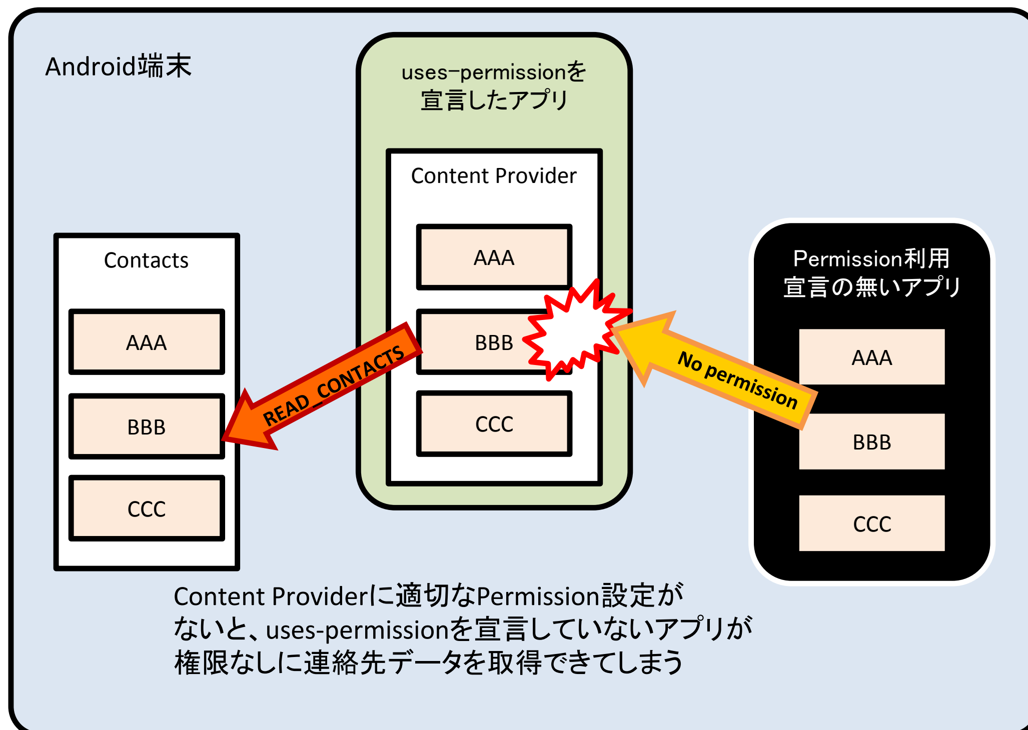


図 5.2-1 Permission を持たないアプリが連絡先情報を取得する

同様の例として、android.permission.CALL_PHONE を利用宣言したアプリが、同 Permission を利用宣言していない他のアプリからの任意の電話番号を受け付け、ユーザーの確認もなくその番号に電話を掛けることができるならば、Permission の再委譲問題がある。

Permission の利用宣言をして得た情報資産・機能資産をほぼそのままの形で他のアプリに二次提供する場合には、提供先アプリに対し同じ Permission を要求するなどして、元の保護水準を維持しなければならない。また情報資産・機能資産の一部分のみを他のアプリに二次提供する場合には、その情報資産・機能資産の一部分が悪用されたときの被害度合に応じた適切な保護が必要である。たとえば前述と同様に同じ Permission を要求したり、ユーザーへ利用許諾を確認したり、「4.1.1.1 非公開 Activity を作る・利用する」「4.1.1.4 自社限定 Activity を作る・利用する」などを利用して対象アプリの制限を設けるなどの保護施策がある。

このような再委譲問題は Permission に限ったことではない。Android アプリでは、アプリに必要な情報･機能を他のアプリやネットワーク・記憶媒体から調達することが一般に行われている。提供元が Android アプリであれば Permission、ネットワークであればログイン、記憶媒体であればアクセス制限といったように、それぞれ調達する際に必要な権限や制限が存在することも多い。こうして調達した情報や機能をその所有者であるユーザーから二次的に他のアプリに提供したり、ネットワークや記憶媒体に転送する際には、ユーザーの意図に反した利用がないように慎重に検討してアプリに対策を施すべきである。必要に応じて、Permission の例と同様に提供先に対して権限の要求や使用の制限を行わなければならない。ユーザーへの利用許諾もその一環である。

以下では、READ_CONTACTS Permission を利用して連絡先 DB から一覧を取得したアプリが、情報提供先のアプリに対して同じ READ_CONTACTS Permission を要求する例を示す。

 All rights reserved © Japan Smartphone Security Association.

ポイント

1. Manifest で提供元と同じ Permission を要求する

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

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

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

        <provider
            android:name=".TransferPermissionContentProvider"
            <!-- ★ポイント 1★ Manifest で提供元と同じ Permission を要求する -->
            android:authorities="org.jssec.android.permission.transferpermission"
            android:enabled="true"
            android:exported="true"
            android:readPermission="android.permission.READ_CONTACTS" >
        </provider>
    </application>

</manifest>
```

アプリが複数の Permission を要求する必要がある場合は、上記の方法では解決することができない。ソースコード上で Context#checkCallingPermission()や PackageManager#checkPermission()を使用して、呼び出し元のアプリが Manifest ですべての Permission の利用宣言を行っているかどうかを確認することになる。

Activity の場合
```
public void onCreate(Bundle savedInstanceState) {
...(Snip)
      PackageManager mgr = getPackageManager();
      if (mgr.checkPermission("android.permission.READ_CONTACTS") == PackageManager.PERMISSION_GRANTED
       && mgr.checkPermission("android.permission.WRITE_CONTACTS") == PackageManager.PERMISSION_GRANTED) {
               // 呼び出し元が正しく Permission を利用宣言していた時の処理
               return;
      }
      finish();
}
```

All rights reserved © Japan Smartphone Security Association.

5.3. Account Manager で独自アカウントを追加する

Account Manager とは様々なサービスとの連携に使用するアカウント情報を統一して管理(登録・保存・利用)するためのフレームワークを提供する機能である。
実際に各サービスのアカウントを操作するには、それぞれに Account Authenticator を用意する必要があるが、Google や Facebook などの主要なサービスの Authenticator は予めインストールされている端末も多い。

Account Manager の最も大きな利点は、アプリと各サービスとの連携を考えた際に、ユーザー認証やそれに伴うユーザーID やパスワードの管理など面倒な処理の実装を Account Manager と各 Authenticator に任せることができる点だ。また、自動ログイン実現に伴うアカウント情報の保存など、センシティブな情報の扱いに関してもアプリごとに個別に実装する必要がない。

ただし、アプリが新しいサービスのアカウント管理機能を端末に加えるために独自の Authenticator を実装する場合には、セキュリティ上で考慮が必要なポイントがいくつかある。

5.3.1. サンプルコード

Authenticator の実装例をサンプルコードとして示す。

ポイント:

1. ログイン情報入力画面の Activity は非公開 Activity(exported=false)とする
2. ログイン情報入力画面の Activity を呼び出す Intent には、明示的にクラスを指定する
3. Authenticator の getAuthTokenLabel()を適切に実装する
4. サーバーとの通信は HTTPS で行う
5. Authenticator の追加したアカウントのパスワード情報を利用するアプリは、Authenticator と同じ鍵で署名する

Authenticator の定義

authenticator.xml

```
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="jssecAccountType"
    android:icon="@drawable/ic_launcher"
    android:smallIcon="@drawable/ic_launcher"
    android:label="@string/label"
/>
```

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.account.authenticatoruser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
```

 All rights reserved © Japan Smartphone Security Association.

```
        android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />

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

</manifest>
```

Authenticator の実装

AccountAuthenticator.java

```
package org.jssec.android.account.authenticator;

import org.jssec.android.account.webservice.WebService;
import android.accounts.AbstractAccountAuthenticator;
import android.accounts.Account;
import android.accounts.AccountAuthenticatorResponse;
import android.accounts.AccountManager;
import android.accounts.NetworkErrorException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;

public class AccountAuthenticator extends AbstractAccountAuthenticator {

    private final Context mContext;
    private static final String JSSEC_ACCOUNT_TYPE = "jssecAccountType";
    private static final String JSSEC_AUTH_TYPE_LABEL = "JSSEC Web Service";

    public AccountAuthenticator(Context context) {
        super(context);
        mContext = context;
    }

    @Override
    public Bundle addAccount(AccountAuthenticatorResponse response, String accountType, String authTokenType, Str
ing[] requiredFeatures, Bundle options)
            throws NetworkErrorException {

        // AuthenticatorActivity に intent を送信
        // ★ポイント 2★ ログイン情報入力画面の Activity を呼び出す Intent には、明示的にクラスを指定する
        final Intent intent = new Intent(mContext, AuthenticatorActivity.class);
        intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE, response);
        final Bundle bundle = new Bundle();
        bundle.putParcelable(AccountManager.KEY_INTENT, intent);
```

All rights reserved © Japan Smartphone Security Association.

```
        return bundle;
    }

    @Override
    public String getAuthTokenLabel(String authTokenType) {
        // ★ポイント 3★ Authenticator の getAuthTokenLabel メソッドを適切に実装する
        return JSSEC_AUTH_TYPE_LABEL;
    }

    @Override
    public Bundle getAuthToken(AccountAuthenticatorResponse response, Account account, String authTokenType, Bund
le options)
            throws NetworkErrorException {

        // AccountManager に登録がある場合
        final AccountManager am = AccountManager.get(mContext);
        final String password = am.getPassword(account);
        if (password != null) {
            WebService web = new WebService();
            final String authToken = web.authenticate(account.name, password);
            if (!TextUtils.isEmpty(authToken)) {
                final Bundle result = new Bundle();
                result.putString(AccountManager.KEY_ACCOUNT_NAME, account.name);
                result.putString(AccountManager.KEY_ACCOUNT_TYPE, JSSEC_ACCOUNT_TYPE);
                result.putString(AccountManager.KEY_AUTHTOKEN, authToken);
                return result;
            }
        }

        // AccountManager に(適切な)登録が無い場合、AuthenticatorActivity に intent を送信
        // ★ポイント 2★ ログイン情報入力画面の Activity を呼び出す Intent には、明示的にクラスを指定する
        final Intent intent = new Intent(mContext, AuthenticatorActivity.class);
        intent.putExtra(AuthenticatorActivity.PARAM_USERNAME, account.name);
        intent.putExtra(AuthenticatorActivity.PARAM_AUTHTOKEN_TYPE, authTokenType);
        intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE, response);
        final Bundle bundle = new Bundle();
        bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        return bundle;
    }

    @Override
    public Bundle confirmCredentials(AccountAuthenticatorResponse response, Account account, Bundle options)
            throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle editProperties(AccountAuthenticatorResponse response, String accountType) {
        return null;
    }

    @Override
    public Bundle updateCredentials(AccountAuthenticatorResponse response, Account account, String authTokenType,
 Bundle options)
            throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle hasFeatures(AccountAuthenticatorResponse response, Account account, String[] features)
```

 All rights reserved © Japan Smartphone Security Association.

```
            throws NetworkErrorException {
        // このアプリでは呼び出されない想定なので、常に false を返す
        final Bundle result = new Bundle();
        result.putBoolean(AccountManager.KEY_BOOLEAN_RESULT, false);
        return result;
    }
}
```

AuthenticationService.java

```
package org.jssec.android.account.authenticator;

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

public class AuthenticationService extends Service {

    private AccountAuthenticator mAuthenticator;

    @Override
    public void onCreate() {
        mAuthenticator = new AccountAuthenticator(this);
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mAuthenticator.getIBinder();
    }
}
```

AuthenticatorActivity.java

```
package org.jssec.android.account.authenticator;

import org.jssec.android.account.webservice.WebService;
import android.accounts.Account;
import android.accounts.AccountAuthenticatorActivity;
import android.accounts.AccountManager;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.view.Window;
import android.widget.EditText;

public class AuthenticatorActivity extends AccountAuthenticatorActivity{

    public static final String PARAM_USERNAME = "usernames";
    public static final String PARAM_AUTHTOKEN_TYPE = "authtokenType";
    private static final String JSSEC_ACCOUNT_TYPE = "jssecAccountType";
    private AccountManager mAccountManager;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        mAccountManager = AccountManager.get(this);

        // sign-in 画面を表示する
```

All rights reserved © Japan Smartphone Security Association.

```
        requestWindowFeature(Window.FEATURE_LEFT_ICON);
        setContentView(R.layout.login_activity);
        getWindow().setFeatureDrawableResource(
                Window.FEATURE_LEFT_ICON, android.R.drawable.ic_dialog_alert);
    }

    /**
     * sign-in ボタン押下時に呼ばれる。
     */
    public void handleLogin(View view) {

        // 入力値を取得
        EditText mUsernameEdit = (EditText) findViewById(R.id.username_edit);
        String mUsername = mUsernameEdit.getText().toString();
        EditText mPasswordEdit = (EditText) findViewById(R.id.password_edit);
        String mPassword = mPasswordEdit.getText().toString();

        if (TextUtils.isEmpty(mUsername) || TextUtils.isEmpty(mPassword)) {
            // 入力不正時の処理
        }

        // web サービスに接続し、認証機能を呼び出す
        WebService web = new WebService();
        String authToken = web.authenticate(mUsername, mPassword);
        if (TextUtils.isEmpty(authToken)) {
            // ログイン認証失敗
            // 認証失敗時の処理
        }

        // 認証済みのアカウントを Account Manager に追加する
        final Account account = new Account(mUsername, JSSEC_ACCOUNT_TYPE);
        mAccountManager.addAccountExplicitly(account, mPassword, null);

        // Authenticator に結果を返す
        finishLogin(mUsername, JSSEC_ACCOUNT_TYPE);
    }

    private void finishLogin(String username, String accountType) {
        // Authenticator に結果を返す
        final Intent intent = new Intent();
        intent.putExtra(AccountManager.KEY_ACCOUNT_NAME, username);
        intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE, accountType);
        setAccountAuthenticatorResult(intent.getExtras());
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

本記事の対象は Web サービスを想定しているが、サンプルのため以下の簡易的なもので代用する。

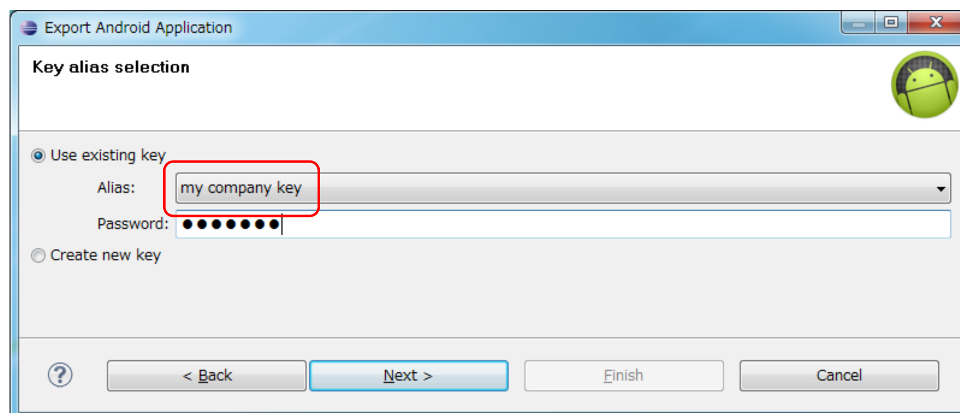
WebService.java

```
package org.jssec.android.account.webservice;

public class WebService {

    /**
     * web サービスの認証機能に接続する想定
```

 All rights reserved © Japan Smartphone Security Association.

```
     * @param username
     * @param password
     * @return トークンを返す
     */
    public String authenticate(String username, String password) {
        // ★ポイント 4★ サーバーとの通信は HTTPS で行う
        // 実際には、サーバーとの通信処理を実装するが、 サンプルにつき割愛
        return getAuthToken(username, password);
    }

    private String getAuthToken(String username, String password) {
        // 実際にはサーバーから、ユニーク性と推測不可能性を保証された値を取得するが
        // サンプルにつき、通信は行わずに固定値を返す
        return "token";
    }
}
```

★ポイント 5★ Authenticator の追加したアカウントのパスワード情報を利用するアプリは、Authenticator と同じ鍵で署名する



All rights reserved © Japan Smartphone Security Association.

5.3.2. ルールブック

Authenticator を実装する際には以下のルールを守ること。

1.	ログイン情報入力画面 Activity は非公開 Activity(exported=false)とする	（必須）
2.	ログイン情報入力画面 Activity を呼び出す Intent には、明示的にクラスを指定する	（必須）
3.	Authenticator の getAuthTokenLabel()を適切に実装する	（必須）
4.	Authenticator の追加したアカウントのパスワード情報を利用するアプリは、Authenticator と同じ鍵で署名する	（必須）

5.3.2.1. ログイン情報入力画面 Activity は非公開 Activity(exported=false)とする （必須）

ログイン情報入力画面 Activity は、Authenticator においてユーザーにユーザーID やパスワードの入力を要求する際に表示される Activity である。この Activity の呼び出しは、Authenticator クラスで実装した addAccount() において初期化した Intent を用いることを想定している。しかし、Intent に適切なパラメータを与えれば、誰でもこの Activityを呼び出せる可能性があり、Authenticatorの実装によっては不正にアカウントを追加できてしまう。そのため、不意に他のアプリから呼び出したりされないように、Activity の公開属性を exported=false とするべきである。また、Activity に Intent Filter タグを定義してもいけない。こうすることで、Activity の呼び出し元を system 権限のアプリか同じ UID を共有するアプリに限定することができる。

なお、端末の「設定」→「アカウントの追加」で呼び出される画面は system 権限で動くアプリなので、すべての Authenticator が持つログイン情報入力画面 Activity を呼び出すことが可能となっている。

「4.1.1.1 非公開 Activity を作る・利用する」も参照のこと。

5.3.2.2. ログイン情報入力画面 Activity を呼び出す Intent には、明示的にクラスを指定する （必須）

Authenticator で実装する addAccount()をはじめ getAuthToken()や updateCredentials()などは、ユーザーの入力が必要な場合には、ログイン情報入力画面 Activity 呼び出し用の Intent を AccountManager.KEY_INTENT キーと対で Bundle に格納して返す。この時、Activity を呼び出す Intent を暗黙的 Intent にすると、意図せず他アプリ(マルウェア)が持つ Activity を呼び出してしまう可能性が生じてしまう。よって、安全のために Activity を呼び出す Intent には明示的な Intent を利用するべきである。

「4.1.1.1 非公開 Activity を作る・利用する」も参照のこと。

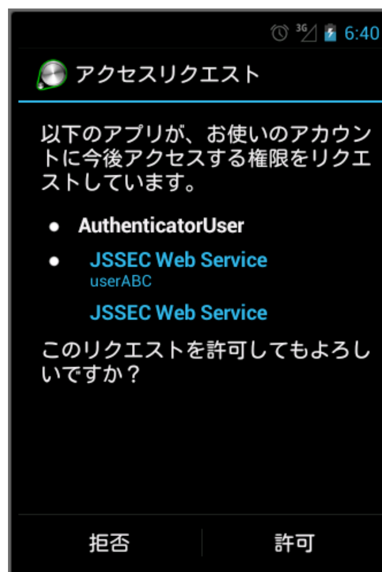
5.3.2.3. Authenticator の getAuthTokenLabel()を適切に実装する （必須）

独自のアカウントタイプを提供するには、AbstractAccountAuthenticator クラスを派生して必要なメソッドをオーバーライドする。実装した Component を含むパッケージを端末にインストールすることで、ユーザーは端末の「設定」→「アカウントの追加」から独自サービスが定義するアカウントタイプでアカウントを追加できるようになる。この際入力したユーザーID とパスワードは、Account Manager が管理するデータベースに保存される。

 All rights reserved © Japan Smartphone Security Association.

保存されたアカウント情報にアクセスするためには、アプリを Authenticator のパッケージと同じ署名鍵で署名し、適切なPermissionを利用宣言する必要がある。署名鍵の異なるアプリもアクセス可能であるが取得できる情報には制限があり、パスワード等の取得は不可能である。そのため、署名鍵の異なる第三者アプリがログインした(認証された)状態でサービスと通信するためには、AccountManager#getAuthToken()で取得した認証トークンを使うことになる。

ここで、第三者アプリに AccountManager#getAuthToken()が呼ばれた時の挙動は、独自 Authenticator でオーバーライドする getAuthTokenLabel()メソッドの実装によって変わる。getAuthTokenLabel()の実装が null でない文字列を返す場合は、トークンの使用許諾を求めるダイアログが表示され、ユーザーによって使用の可否が判断される。ユーザーが許可した場合のみアプリはトークンを使用することができる。逆にメソッドが null を返す場合にはダイアログは表示されず、端末ユーザーの許可無しに認証トークンを使用することができる。



このように、一般のアプリがトークンを取得する際に、ユーザーに許可を求めることで認証トークンの不正使用のリスクを減らすことができる。そのため、getAuthTokenLabel()の実装では null を返さずに適切な文字列を返すように実装するべきである。

なお、第三者アプリが認証トークンを取得するためには USE_CREDENTIALS Permission も必要となる。

5.3.2.4. Authenticator の追加したアカウントのパスワード情報を利用するアプリは、Authenticator と同じ鍵で署名する （必須）

AccountManager クラスの各メソッドを利用するためには、アプリの Manifest にそれぞれ適正な Permission の利用宣言をする必要がある。Permission とメソッドの対応表は「5.3.3.1Account Manager の利用と Permission」に示す。

ここで、AUTHENTICATE_ACCOUNTS Permission が必要なメソッド群を使う場合には Permission に加えてパッ

All rights reserved © Japan Smartphone Security Association.

ケージの署名鍵に関する制限が設けられている。具体的には、Authenticator を提供するパッケージの署名に使う鍵とメソッドを使うアプリのパッケージの署名に使う鍵が同じでなければならない。そのため、Authenticator 以外に AUTHENTICATE_ACCOUNTS Permission が必要なメソッド群を使うアプリを配布する際には、Authenticator と同じ鍵で署名を施すことになる。

Eclipse での開発の際には設定した署名鍵が固定で使われるため、鍵のことを意識せずに Permission だけで実装や動作確認が出来てしまう。特にアプリによって署名鍵を使い分けている開発者は、この制限を考慮してアプリに使う鍵を選定する必要があるので注意をすること。

 All rights reserved © Japan Smartphone Security Association.

5.3.3. アドバンスト

5.3.3.1. Account Manager の利用と Permission

Account Manager が提供している機能を利用するには、機能毎に適正な Permission の利用宣言が必要である。

Account Manager が提供する機能と必要な Permission の対応表を以下に示す。

表 5.3.3-1 Account Manager の機能と Permission

	Account Manager が提供する機能	
Permission	メソッド	説明
AUTHENTICATE_ACCOUNTS (Authenticator と同じ鍵で署名された Package のみ利用可能)	getPassword()	パスワードの取得
	getUserData()	ユーザー情報の取得
	addAccountExplicitly()	アカウントの DB への追加
	peekAuthToken()	キャッシュされたトークンの取得
	setAuthToken()	認証トークンの登録
	setPassword()	パスワードの変更
	setUserData()	ユーザー情報の設定
GET_ACCOUNTS	getAccounts()	すべてのアカウントの一覧取得
	getAccountsByType()	アカウントタイプが同じアカウントの一覧取得
	getAccountsByTypeAndFeatures()	指定した機能を持ったアカウントの一覧取得
	addOnAccountsUpdatedListener()	イベントリスナーの登録
	hasFeatures()	指定した機能の有無
MANAGE_ACCOUNTS	getAuthTokenByFeatures()	指定した機能を持つアカウントの認証トークンの取得
	addAccount()	ユーザーへのアカウント追加要請
	removeAccount()	アカウントの削除
	clearPassword()	パスワードの初期化
	updateCredentials()	ユーザーへのパスワード変更要請
	editProperties()	Authenticator の設定変更
	confirmCredentials()	ユーザーへのパスワード再入力要請
USE_CREDENTIALS	getAuthToken()	認証トークンの取得
	blockingGetAuthToken()	認証トークンの取得
MANAGE_ACCOUNTS または USE_CREDENTIALS	invalidateAuthToken()	キャッシュされたトークンの削除

Account Manager から取得するデータはセンシティブな情報であるため、漏洩や不正利用などのリスクを減らすように扱いには十分注意すること。

5.3.3.2. 重要な処理の前には、ユーザーにパスワードの再入力を要求する

商品購入(決済処理)やユーザー情報変更などの重要な処理を実行する前に、パスワード入力画面を表示してユーザーにパスワードの再入力を求め、再認証を行うことで、万一認証トークンが第三者に漏洩した場合にもその影響や被害を減少させることが出来る。

もちろん認証トークンの漏洩(推測)・再利用を防止するようにアプリやサービスの設計・実装を行うことが最も重要である。しかし、自責以外にも、使用したライブラリや端末自身に脆弱性があったりするなど、完璧に漏洩が起こらないと保証することは難しい。開発者にできることはその機会を可能な限り減らすということだけである。

このような理由からパスワードの再入力は必要であり、検討すべき重要なセキュリティ対策となる。
ただし、パスワードの再入力を頻繁に行うとアプリのユーザビリティが損なわれるので、トークン漏洩によるユーザーの被害やビジネスインパクトを考慮して、仕様や設計の段階から対策が必要な個所を吟味し、対応する必要がある。

5.3.3.3. 認証に使われるトークンについて

認証トークンはログイン(認証)が成功した際にサービスが生成するアカウント毎にユニークな乱数である。アプリは認証トークンを使うことによって、パスワードを毎回送信することなく、一定期間サービスとのログイン状態を保持することができる。

パスワードも認証トークンも漏洩すれば第三者がユーザーに成りすまして機能を利用されてしまう恐れがある。しかし、認証トークンの場合は適切な有効期限を設けたり、重要な処理の前にパスワードの再入力をユーザーに求めて認証トークンを再発行することでリスクを低減できる。また、認証トークンの有効期限はサーバー側で制御ができるため、万一の場合も簡単に無効にすることができる。パスワードの場合は、サービスがユーザーに提供しているすべての機能にアクセス可能になるため、漏洩した場合には、より詳細な個人情報やプライバシー情報にアクセスしたり、場合によってはパスワード自体を変更されてユーザー本人がサービスにアクセスできなくなるなど、被害が大きくなる可能性が高い。

上記のように、パスワードと比較すれば、認証トークンは漏洩した時の対応を軽減するように設計することが出来る。しかしながら、基本的にはユーザーにしか許可されていない機能にアクセスできるので、許可される機能の重要度によって期間や制限など適切な対策を施すとともに(アプリでの)取り扱いは細心の注意を払って行うべきである。具体的には、ユーザーの許可なしに他のアプリに使用させない(「4.1.2.1 アプリ内でのみ使用する Activity は非公開設定する　　　　（必須）」参照)、サービスとの通信時は暗号化通信で第三者に盗聴されないように保護するなど、設計上のセキュリティ対策を施す必要がある。

 All rights reserved © Japan Smartphone Security Association.

5.3.3.4. 複数の Authenticator で同一のアカウントタイプが定義された場合

端末に同一のアカウントタイプを定義した Authenticator が複数存在する場合、先にインストールされた Authenticator が有効になる。独自定義のアカウントタイプを持った他者の Authenticator が先にインストールされていたら、自分が作成した Authenticator が使われないということである。仮に先にインストールされた Authenticator がマルウェアによる偽装であると、ユーザーが入力したアカウント情報が取られてしまう恐れがある。

このように他者の Authenticator が先に登録され、自前の Authenticator が無効となっている場合への対策は、誰のアプリが AccountManager を使って Authenticator を呼び出すかによって場合分けをすることができる。

まず、「設定」アプリ経由でアカウントを追加するなど、システム経由で Authenticator の呼び出しを行う場合である。この場合、ユーザー操作や Authenticator の呼び出しに全く関与できない。つまり、実装による直接の対策を施すことができない。そのため、マルウェアをインストールしない対策を施すなどユーザーに対して注意喚起する以上には、十分な予防策がないのが現実である。

一方、自前のアプリから AccountManager 経由で Authenticator を呼び出す場合には、呼び出す Authenticator が自前のものかどうかを判定する方法(後述)がある。そのため、他者 Authenticator が呼び出されることが判明した際に、ユーザーにその旨を伝えてアンインストールを促すといった対処を施すことが可能である。

以下に、自前の Authenticator を判定するサンプルコードを示す。

ポイント:

1. 利用する Authenticator の証明書が自社の証明書であることを確認する

AndroidManifest.xml
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.account.authenticatoruser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".UseAuthenticatorActivity"
            android:label="@string/title_activity_use_authenticator" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

All rights reserved © Japan Smartphone Security Association.

```
        </activity>
    </application>

</manifest>
```

UseAuthenticatorActivity.java

```
package org.jssec.android.account.authenticatoruser;

import java.io.IOException;

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

import android.accounts.Account;
import android.accounts.AccountManager;
import android.accounts.AuthenticatorDescription;
import android.accounts.AuthenticatorException;
import android.accounts.OperationCanceledException;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class UseAuthenticatorActivity extends Activity {

    // 利用する Authenticator の情報
    private static final String JSSEC_ACCOUNT_TYPE = "jssecAccountType";
    private static final String TARGET_PACKAGE = "org.jssec.android.account.authenticator";

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

    /**
     * ボタン押下時に呼ばれる
     */
    public void useAuthenticator(View view) {
        try {
            AccountManager am = AccountManager.get(this);

            // Authenticator を判定する
            // Account Type が一致する Authenticator を取得する
            AuthenticatorDescription[] adArr = am.getAuthenticatorTypes();
            String packageName = null;
            for (AuthenticatorDescription ad: adArr) {
                if (JSSEC_ACCOUNT_TYPE.equals(ad.type)) {
                    packageName = ad.packageName;
                    break;
                }
            }
            String message, info;
            if (packageName != null) {
                info = "<info>¥n" + "アカウントタイプ：" + JSSEC_ACCOUNT_TYPE;
                info = info + "¥n" + "現在有効な Authenticator パッケージ名：¥n" + packageName;
                info = info + "¥n" + "自社 Authenticator パッケージ名：¥n" + TARGET_PACKAGE;
```

 All rights reserved © Japan Smartphone Security Association.

```
                if (TARGET_PACKAGE.equals(packageName)) {

                    // ★ポイント 1★ 利用する Authenticator 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))) {
                        message = "Authenticator を利用できません(証明書不一致)";
                    } else {
                        message = "Authenticator が有効です";

                        // token を取得する
                        Account[] accounts = am.getAccountsByType(JSSEC_ACCOUNT_TYPE);
                        for (Account account: accounts) {
                            String token =  am.blockingGetAuthToken(account, JSSEC_ACCOUNT_TYPE, false);
                            message = message + "¥n" + account.name + "さんのトークンを取得しました⇒" + token;
                        }
                    }
                } else {
                    message = "Authenticator を利用できません(パッケージ不一致)";
                }
            } else {
                message = "Authenticator が存在しません";
                info = "";
            }

            TextView textView =(TextView) findViewById(R.id.textView1);
            textView.setText(message + "¥n¥n" + info);

        } catch (IOException e) {
            // 例外処理
        } catch (AuthenticatorException e) {
            // 例外処理
        } catch (OperationCanceledException e) {
            // 例外処理
        }
    }

    // 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;
    }
}
```

PkgCert.java

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

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import android.content.pm.Signature;
import android.content.Context;
```

```
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
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;    // 複数署名は扱わない
            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 StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

5.3.3.5. パスワードの保存について

Account Manager はアカウント追加の際にユーザーID やパスワードをデータベースに保存するが、データベースファイルは system 権限にアクセスが制限されているため、ユーザーや一般権限のアプリが直接アクセスすることはできない。また、AccountManager の提供する getPassword メソッドの使用は Permission に加えて Authenticator と同じ鍵の署名が必要であり、Authenticator の開発元以外のアプリからはパスワードを取得する方法が提供されていないので、本ガイドのスコープではパスワードは安全に保護されていると考えることができ、サンプルコードでも同様の視点で実装例を示している。

 All rights reserved © Japan Smartphone Security Association.

一方で、データベースに保存されるデータはすべて平文として扱われる。Account Manager はファイルのアクセス制限以外には、パスワードなどのセンシティブなデータを保護する仕組みは持っておらず、データの特別な保護は各 Authenticator の実装に委ねられている。このため、端末の system 権限や root 権限を奪取されてもパスワードをはじめセンシティブな情報を守る必要のある場合には、Authenticator で暗号化するなどの保護対策を施すことになる。

パスワードの保護は、「パスワードを保存しない」方法から高度な「暗号技術」「耐タンパ技術」を使った方法まで、保護の度合(守るべき資産の価値)やサービスとの連携、仕様上の制限など含めて取り得る対策が変わるため、本ガイドではスコープ外とする。必要に応じて Android に詳しいセキュリティ専門家に相談することをお勧めする。

All rights reserved © Japan Smartphone Security Association.

6. 難しい問題

Android には OS の仕様や OS が提供する機能の仕様上、アプリの実装でセキュリティを担保するのが困難な問題が存在する。これらの機能は悪意を持った第三者に悪用されたり、ユーザーが注意せずに利用したりすることで、情報漏洩を始めセキュリティ上の問題に繋がってしまう危険性を常に抱えている。この章ではそのような機能に対して、開発者が取りうるリスク削減策などを提示しながら注意喚起が必要な話題を記事として取り上げる。

6.1. Clipboard から情報漏洩する危険性

コピー＆ペーストはユーザーが普段から何気なく使っている機能であろう。例えば、この機能を使って、メールや Web ページで気になった情報や忘れたら困る情報をメモ帳に残しておいたり、設定したパスワードを忘れないようにメモ帳に保存しておき、必要な時にコピー＆ペーストして使うというユーザーは少なからず存在する。これらは一見何気ない行為であるが、実はユーザーの扱う情報が盗まれるという危険が潜んでいる。

これには Android のコピー＆ペーストの仕組みが関係している。ユーザーやアプリによってコピーされた情報は、一旦 Clipboard と呼ばれるバッファに格納される。ユーザーやアプリによってペーストされたときに、この Clipboard の内容が各アプリに再配布されるわけである。この Clipboard に情報漏洩に結び付く危険性がある。Android 端末の仕様では、Clipboard の実体は端末に 1 つであり、ClipboardManager を利用することで、どのアプリからでも常時 Clipboard の中身が取得できるようになっているからである。このことは、ユーザーがコピー・カットした情報は全て悪意あるアプリに対して筒抜けになることを意味している。

よって、アプリ開発者は、この Android の仕様を考慮しながら情報漏洩の可能性を最小限に抑える対策を講じなくてはならない。

6.1.1. サンプルコード

Clipboard から情報漏洩する可能性を抑える対策には、大きく分けて次の 2 つが考えられる。

1. 他アプリから自アプリへコピーする際の対策
2. 自アプリから他アプリへコピーする際の対策

最初に、1.について説明する。ここでは、ユーザーがメモ帳や Web ブラウザ、メーラーアプリなど他アプリから文字列をコピーし、それを自アプリの EditText に貼り付けるシナリオを想定している。結論だけを言ってしまうと、このシナリオでコピー・カットによってセンシティブな情報が漏洩してしまうことを防ぐ根本的な対策は存在しない。第三者アプリのコピー機能を制御するような機能が Android にはないからだ。

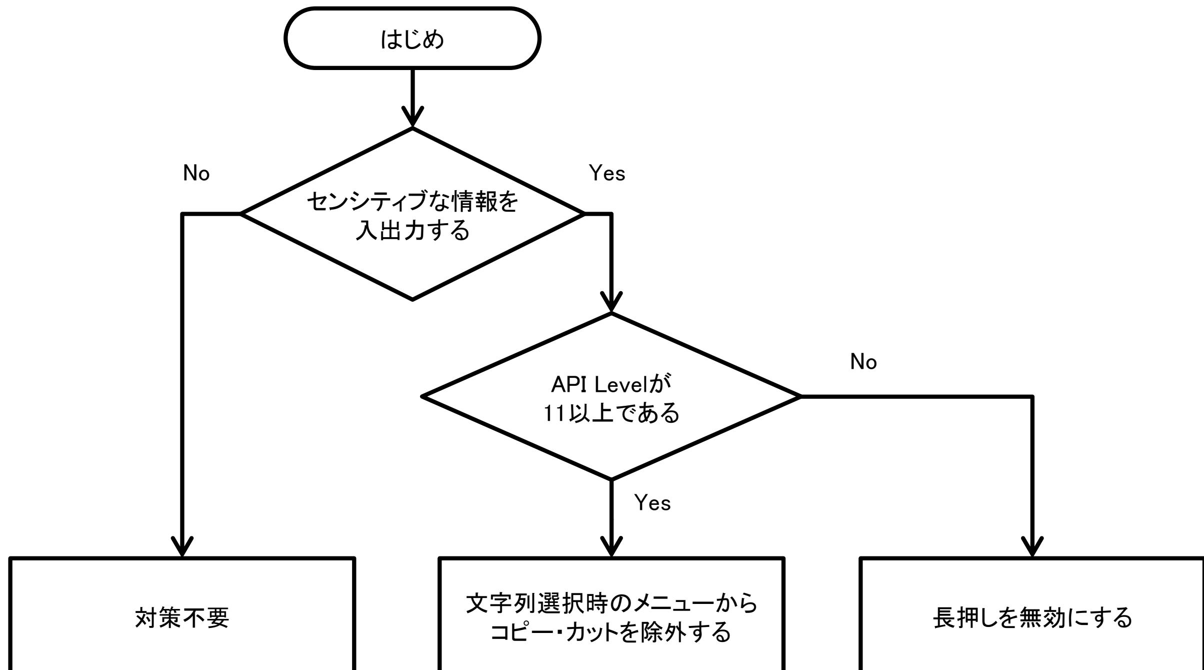
よって、1.についてはセンシティブな情報をコピー・カットする危険性をユーザーに説明し、行為自体を減らしていく啓発活動を継続的に行っていくしか対策はない。

次に、2.を説明する。ここでは、自アプリが表示している情報がユーザーによってコピーされるシナリオを想定する。こ

 All rights reserved © Japan Smartphone Security Association.

の場合、漏洩に対する確実な対策は、View(TextView, EditText など)からのコピー・カットを禁止にすることである。個人情報などセンシティブな情報が入力あるいは出力される View にコピー・カット機能がなければ、自アプリからの Clipboard を介した情報の漏洩もないからだ。

コピー・カットを禁止する方法は API Level などによっていくつかある。下図の判定フローに従って、実施すべき対策を決定することができる。



ただし、入力タイプ(Input Type)が下記のいずれかに固定されている(=動的に変更されない)EditText に関しては、デフォルトでコピー・カットが無効なので、判定の結果に関わらず対策不要である。

- InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD
- InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_WEB_PASSWORD
- InputType.TYPE_CLASS_NUMBER | InputType.TYPE_NUMBER_VARIATION_PASSWORD

以下で、各対策の詳細を説明し、サンプルコードを示す。

6.1.1.1. 文字列選択時のメニューからコピー・カットを削除する

Android 3.0(API Level 11)以上では、TextView.setCustomSelectionActionModeCallback()メソッドによって、文字列選択時のメニューをカスタマイズできるようになった。これを用いて、文字列選択時のメニューからコピー・カットのアイテムを削除すれば、ユーザーが文字列をコピー・カットすることはできなくなる。

以下、EditText の文字列選択時のメニューからコピー・カットの項目を削除するサンプルコードを示す。

All rights reserved © Japan Smartphone Security Association.

ポイント：

1. 文字列選択時のメニューから android.R.id.copy を削除する。
2. 文字列選択時のメニューから android.R.id.cut を削除する。

UncopyableActivity.java

```
package org.jssec.android.clipboard.leakage;

import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.view.ActionMode;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;

public class UncopyableActivity extends Activity {
    private EditText copyableEdit;
    private EditText uncopyableEdit;

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

        copyableEdit = (EditText) findViewById(R.id.copyable_edit);
        uncopyableEdit = (EditText) findViewById(R.id.uncopyable_edit);
        // API Level 11 以上であれば、setCustomSelectionActionModeCallback メソッドにより、
        // 文字列選択時のメニューをカスタマイズすることができる。
        uncopyableEdit.setCustomSelectionActionModeCallback(actionModeCallback);
    }

    private ActionMode.Callback actionModeCallback = new ActionMode.Callback() {
        public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
            return false;
        }

        public void onDestroyActionMode(ActionMode mode) {
        }

        public boolean onCreateActionMode(ActionMode mode, Menu menu) {
            // ★ポイント 1★ 文字列選択時のメニューから android.R.id.copy を削除する。
            MenuItem itemCopy = menu.findItem(android.R.id.copy);
            if (itemCopy != null) {
                menu.removeItem(android.R.id.copy);
            }
            // ★ポイント 2★ 文字列選択時のメニューから android.R.id.cut 削除する。
            MenuItem itemCut = menu.findItem(android.R.id.cut);
            if (itemCut != null) {
                menu.removeItem(android.R.id.cut);
            }
            return true;
        }

        public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
            return false;
        }
    };
```

 All rights reserved © Japan Smartphone Security Association.

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.uncopyable, menu);
        return true;
    }


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

6.1.1.2. View の長押し(Long Click)を無効にする

Android 3.0(API Level 11)未満では、TextView.setCustomSelectionActionModeCallback()メソッドは利用できない。この場合、コピー・カットを禁止する最も簡単な方法は、View の長押し(Long Click)を無効にすることである。View の長押し無効化はレイアウトの xml ファイルで指定することができる。

ポイント:

1. コピー･カットを禁止する View は android:longClickable を false にする。

unlongclickable.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/unlongclickable_description" />

    <!-- コピー・カットを禁止する EditText -->
    <!-- ★ポイント 1★ コピー・カットを禁止する View は android:longClickable を false にする。 -->
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:longClickable="false"
        android:hint="@string/unlongclickable_hint" />
</LinearLayout>
```

6.1.2. ルールブック

自アプリから他アプリへのセンシティブな情報のコピーが発生する可能性がある場合は、以下のルールを守ること。

1.	Android 3.0(API Level 11)以降では、文字列選択時のメニューからコピー・カットを除外する	(必須)
2.	Android 3.0(API Level 11)未満では、View の長押し(Long Click)を禁止する	(必須)

6.1.2.1. Android 3.0(API Level 11)以降では、文字列選択時のメニューからコピー・カットを除外する (必須)

アプリがセンシティブな情報を表示する View を持っている場合、それが EditText のようにコピー・カットが可能な View ならば、Clipboard を介してその情報が漏洩してしまう可能性がある。そのため、センシティブな情報を表示する View はコピー・カットを無効にしておかなければならない。

Android 3.0(API Level 11)以降では、setCustomSelectionActionModeCallback()メソッドによってコピー・カットを無効にしておくことで実現ができる。

「6.1.3.1 ルール適用の際の注意」も参照のこと。

6.1.2.2. Android 3.0(API Level 11)未満では、View の長押し(Long Click)を禁止する (必須)

アプリがセンシティブな情報を表示する View を持っている場合、それが EditText のようなコピー・カットが可能な View ならば、Clipboard を介してその情報が漏洩してしまう可能性がある。そのため、センシティブな情報を表示する View はコピー・カットを無効にしておかなければならない。

Android 3.0(API Level 11)未満の場合、View の長押し(Long Click)を無効にしておくことで実現できる。

「6.1.3.1 ルール適用の際の注意」も参照のこと。

 All rights reserved © Japan Smartphone Security Association.

6.1.3. アドバンスト

6.1.3.1. ルール適用の際の注意

TextView はデフォルトでは文字列選択不可であるため、通常は対策不要であるが、アプリの仕様によってはコピーを可能にする場合もある。Android3.0(API Level 11)以降では、TextView.setTextIsSelectable()メソッドを使うことで、文字列の選択可否とコピー可否を動的に設定できるようになった。TextView をコピー可能とする場合は、その TextView にセンシティブな情報が表示される可能性がないかよく検討し、その可能性があるのであれば、コピー可にすべきでない。

また、「6.1.1 サンプルコード」の判定フローにも記載されているように、パスワードの入力を想定した入力タイプ(InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD など)の EditText については、デフォルトで文字列のコピーが禁止されているため通常は対策不要である。しかし、「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）」に記載したように「パスワードを平文表示する」オプションを用意している場合は、パスワード平文表示の際に入力タイプが変化し、コピー・カットが有効になってしまうので、同様の対策が必要である。

なお、ルールを適用する際には、ユーザビリティの面も考慮する必要があるだろう。例えば、ユーザーが自由にテキストを入力できる View の場合、センシティブな情報が入力される「可能性がゼロでない」からといってコピー・カットを無効にしてしまったら、ユーザーの使い勝手が悪くなるだろう。もちろん、重要度の高い情報を入出力する View やセンシティブな情報を単独で入力するような View にはルールを無条件で適用するべきであるが、それ以外の View を扱う場合は、次のことを考慮しながら対応を考えると良い。

- センシティブな情報の入力や表示を行う専用のコンポーネントを用意できないか
- 連携先(ペースト先)アプリが分かっている場合は、他の方法で情報を送信できないか
- アプリでユーザーに入出力に関する注意喚起ができないか
- 本当にその View が必要か

Android OS の Clipboard と ClipboardManager の仕様にセキュリティに対する考慮がされていないことが情報漏洩の可能性を生む根本的な要因ではあるが、アプリ開発者は、ユーザー保護やユーザビリティ、提供する機能など様々な観点からこうした Clipboard の仕様に対して対応し、質の高いアプリを作成する必要がある。

6.1.3.2. Clipboard に格納されている情報の操作

6.1.で述べたように、ClipboardManager を利用することでアプリから Clipboard に格納された情報を操作することができる。また、ClipboardManager の利用には特別な Permission を設定する必要が無いため、アプリはユーザーに知られることなく ClipboardManager を利用できる。

ここでは、参考までに API Level 11 以降での実現方法について述べる。

All rights reserved © Japan Smartphone Security Association.

Clipboard に格納されている情報(ClipData と呼ぶ)は、ClipboardManager.getPrimaryClip()メソッドによって取得できる。タイミングに関しても、OnPrimaryClipChangedListener を実装して ClipboardManager.addPrimaryClipChangedListener()メソッドで ClipboardManager に登録すれば、ユーザーの操作などにより発生するコピー・カットの度に Listener が呼び出されるので、タイミングを逃すことなく ClipData を取得することができる。ここで Listener の呼び出しは、どのアプリでコピー・カットが発生したかに関係なく行われる。

以下、端末内でコピー・カットが発生する度に ClipData を取得し、Toast で表示する Service のソースコードを示す。下記のような簡単なコードにより Clipboard に格納された情報が筒抜けになってしまうことを実感していただきたい。アプリを実装する際は、少なくとも下記のコードによってセンシティブな情報が取得されてしまうことのないように注意する必要がある。

ClipboardListeningService.java
```java
package org.jssec.android.clipboard;

import android.app.Service;
import android.content.ClipData;
import android.content.ClipboardManager;
import android.content.ClipboardManager.OnPrimaryClipChangedListener;
import android.content.Context;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class ClipboardListeningService extends Service {
    private static final String TAG = "ClipboardListeningService";
    private ClipboardManager mClipboardManager;

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        mClipboardManager = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
        if (mClipboardManager != null) {
            mClipboardManager.addPrimaryClipChangedListener(clipListener);
        } else {
            Log.e(TAG, "ClipboardService の取得に失敗しました。サービスを終了します。");
            this.stopSelf();
        }
    }

    @Override
    public int onStartCommand(Intent intent, int flag, int startId) {
        super.onStartCommand(intent, flag, startId);
        ServiceRunningStatus.setStatus(getApplicationContext(), true);
        return START_STICKY;
    }

    @Override
```

 All rights reserved © Japan Smartphone Security Association.

```
    public void onDestroy() {
        super.onDestroy();
        if (mClipboardManager != null) {
            mClipboardManager.removePrimaryClipChangedListener(clipListener);
            ServiceRunningStatus.setStatus(getApplicationContext(), false);
        }
    }

    private OnPrimaryClipChangedListener clipListener = new OnPrimaryClipChangedListener() {
        public void onPrimaryClipChanged() {
            if (mClipboardManager != null && mClipboardManager.hasPrimaryClip()) {
                ClipData data = mClipboardManager.getPrimaryClip();
                ClipData.Item item = data.getItemAt(0);
                Toast
                    .makeText(
                        getApplicationContext(),
                        "コピーあるいはカットされた文字列:¥n"
                            + item.coerceToText(getApplicationContext()),
                        Toast.LENGTH_SHORT)
                    .show();
            }
        }
    };
}
```

次に、上記 ClipboardListeningService を利用する Activity のソースコードの例を示す。

ClipboardListeningActivity.java

```
package org.jssec.android.clipboard;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

public class ClipboardListeningActivity extends Activity {
    private static final String TAG = "ClipboardListeningActivity";
    private boolean isServiceRunning;

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

        isServiceRunning = ServiceRunningStatus.getStatus(getApplicationContext());
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_clipboard_listening, menu);
        return true;
```

All rights reserved © Japan Smartphone Security Association.

```
    }

    public void onClickStartService(View view) {
        if (view.getId() != R.id.start_service_button) {
            Log.w(TAG, "View ID が不正です");
        } else {
            if (!isServiceRunning) {
                ComponentName cn = startService(
                        new Intent(ClipboardListeningActivity.this, ClipboardListeningService.class));
                if (cn != null) {
                    this.isServiceRunning = true;
                } else {
                    Log.e(TAG, "サービスの起動に失敗しました");
                    Toast.makeText(this, "サービスの起動に失敗しました", Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(this, "サービスは既に起動しています", Toast.LENGTH_SHORT).show();

            }
        }
    }

    public void onClickStopService(View view) {
        if (view.getId() != R.id.stop_service_button) {
            Log.w(TAG, "View ID が不正です");
        } else {
            if (isServiceRunning) {
                boolean res = stopService(
                        new Intent(ClipboardListeningActivity.this, ClipboardListeningService.class));
                if (res) {
                    this.isServiceRunning = false;
                } else {
                    Log.e(TAG, "サービスの停止に失敗しました");
                    Toast.makeText(this, "サービスの停止に失敗しました", Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(this, "サービスは既に停止しています", Toast.LENGTH_SHORT).show();

            }
        }
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

なお、API Level 11 より前の ClipboardManager には、setOnPrimaryClipChangedListener()メソッドのような Listener を登録するメソッドが用意されていない。それゆえ、ユーザーの操作によって発生するコピー・カットの度に ClipData を取得するプログラムを実装することができないが、AlarmManager などを利用して短い時間間隔で

 All rights reserved © Japan Smartphone Security Association.

Clipboard に格納されている情報を繰り返し取得することで、同等の挙動を示すプログラムを実装することができる。また、API Level 11 以前で実装する場合、getPrimaryClip()メソッドは利用できないので、Clipboard に格納されている情報は getText()メソッドによって文字列として取得することになる。ここではサンプルコードは割愛する。

ここまでは、Clipboard に格納された情報を取得する方法について述べたが、Clipboard に新しく情報を格納することも可能である。API Level 11 以上ならば ClipboardManager.setPrimaryClip()メソッドによって、それ以前であれば ClipboardManager.setText()メソッドを用いればよい。

ただし、setPrimaryClip()あるいは setText()メソッドは Clipboard に格納されていた情報を上書きするので、ユーザーが予めコピー・カット操作により格納しておいた情報が失われる可能性がある点に注意が必要である。これらのメソッドを使用して独自のコピー機能あるいはカット機能を提供する場合は、必要に応じて、内容が改変される旨を警告するダイアログを表示するなど、Clipboard に格納されている内容がユーザーの意図しない内容に変更されることのないように設計・実装する必要がある。

All rights reserved © Japan Smartphone Security Association.