第21章

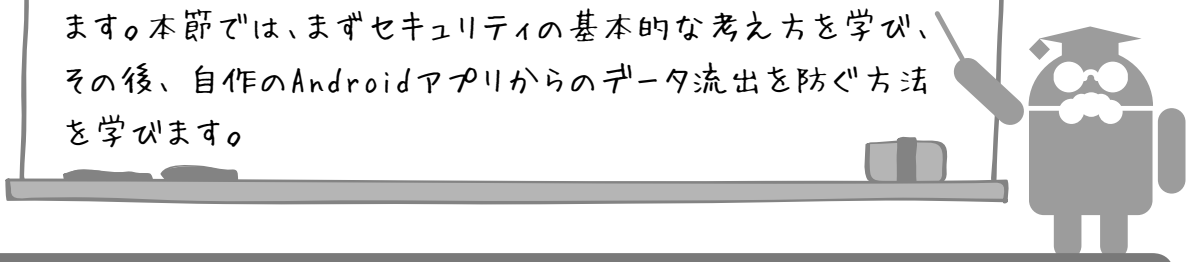
セキュリティ

著：中西良明

21-1 Androidアプリのセキュリティの基本

著：中西良明

開発時点からしっかりとセキュリティを考えておくことで、アプリのリリース後に発生するトラブルのリスクを減らすことができます。本節では、まずセキュリティの基本的な考え方を学び、その後、自作のAndroidアプリからのデータ流出を防ぐ方法を学びます。



この節で学ぶこと

- セキュリティの基本
- Androidアプリ内部データを外部アプリから守る方法

この節で出てくるキーワード一覧

permission
android:exported
暗号
共通鍵暗号
公開鍵暗号

この章で出てくるクラス

Context
SharedPreferences
ContentProvider

21-1-1　アプリのセキュリティとは何か？

セキュリティと聞いたとき、いったいどんなことを考えますか？　ウィルスやトロイの木馬などによって、データを盗まれることや破壊されることに対する防御を思い浮かべるのではないでしょうか。

当然それらも重要なセキュリティの1つです。しかし、アプリを開発するときにセキュリティに気をつけるというのは、アプリの持つデータをマルウェアに盗まれたり破壊されたりしないようにすることだけではありません。アプリ開発においてセキュリティを考えるというのはどういうことかを学ぶ前に、まずはこれまでにあったセキュリティ問題の事例について振り返ります。

アプリのセキュリティ問題事例

Androidアプリにおいて、過去に様々なアプリがセキュリティ問題を起こしてきました。ここではその一部の概要を紹介します。

(1)浮気防止アプリで、彼氏(彼女)のスマートフォンの利用履歴を収集し、自分の端末に対して送信していた
(2)SNSアプリ、チャットアプリなどで、ユーザーの電話帳情報を収集しサーバーに勝手に送信していた
(3)アプリからSDカード(拡張ストレージ)上に暗号化せずにデータを保存していた
(4)データをアプリ外から直接アクセスできないところに置いていたが、データ管理機能のアクセス制限に不備があり、そのデータ管理機能を経由することで誰でもアクセスできる状態になっていた
(5)サーバーとの暗号通信接続手順の不備により、サーバーとの通信を傍受することが可能になっていた
(6)サーバーとの暗号通信接続時に、弱い暗号方式が選ばれていた
(7)Twitterクライアントアプリで、他のアプリから勝手に画像アップロード機能を呼び出せるようになっていた
(8)ログから、ユーザーの個人情報(アカウント、メールアドレスなど)を傍受することが可能になっていた

(1)と(2)は、アプリが意図的に持っていた機能が利用者から不評を買い、社会問題化した事例です。(3)〜(8)は、開発者が意図しない問題がアプリ内にあった事例です。

前者は仕様に、後者は実装にセキュリティ問題があったと言えます。仕様と実装のセキュリティについての考え方を理解しておくことは重要です。

仕様におけるセキュリティ

アプリの「仕様」とは、そのアプリがどういう機能をユーザーに提供するかということです。たとえば、以下のような機能を持ったアプリを考えてみましょう。

- **浮気防止のため、彼氏（彼女）のスマートフォン利用履歴を収集し、自分の端末に対して送信する**
- **スマートフォンを利用する子供がトラブルに巻き込まれないようにするため、監視ソフトをインストールした端末を与える**
  →子ども向けに機能限定されたスマートフォンで、通信のフィルタリングやコミュニケーション相手を制限する
  →会社から従業員に、通信接続先を制限したスマートフォンを与える
- **ユーザーの家族や友人に招待を送るため、電話帳情報を取得してサーバーに対して送る**
  →ユーザーの許可をもらってから情報を収集する
  →ユーザーには知らせず電話帳情報を収集する

これらのアプリ（機能）は問題がないでしょうか。利用履歴の収集や、電話帳情報の取得は、過去の事例では「無断で」おこなうことに問題があるとみなされました。浮気防止の事例では、ユーザーが相手の了承を得た上で、相手の端末にインストールをすることを建前としていました。しかし、実際には相手の同意を得ずにこっそりとインストールする場合が大半であることが想定されたため、ネット上での議論は炎上しました。結果、本格的な開始の前にサービスを終了することとなりました。無断での電話帳収集は、現在では非常に有名になったサービスがユーザー獲得のためにおこなっていた施策の1つです。現在ではそのような動作をしないよう修正されています。

では、ユーザーに対して十分に説明した上で同意を取り、電話帳情報を収集することは問題ないのでしょうか。これについてはグレーゾーンであると考えられています（セキュリティに厳しい方は問題であると考えています）。電話帳情報の収集は、その電話帳を持っている人の同意を得ているとしても、電話帳に含まれる情報は、その電話帳の持ち主のものではなく、電話帳に載せられている人の情報であるためです。

なお、子供用のスマートフォンに監視ソフトをインストールすることや、会社から従業員に、通信接続先を制限したスマートフォンを与えるといったことは、機能的には浮気防止の場合とそれほど違いがありませんが、社会的には問題とされない（されにくい）例であると考えられます。というのも、親子や会社と従業員の間には、管理する側とされる側という関係が存在するためです。

開発に時間や費用をかけたにもかかわらず、アプリやサービスを終了せざるを得

なくなることは非常に大きな無駄となります。よって、アプリのアイデアを考える際に、そのアイデアは法的に問題が無いか、また社会に受け入れられるものかを考えておくことは非常に重要です。

パーミッション宣言

Androidには、アプリがどんな権限を利用するかを宣言する仕組みがあります。それがパーミッションです。これはアプリがどういう仕様を持っているかを、間接的な形で宣言しているとも考えられます。

アプリ開発の演習の中で、「AndroidManifest.xml」に以下のように記述したことがあるはずです。

リスト1

```
<uses-permission android:name="android.permission.INTERNET" />
```

「このアプリはインターネットとの通信をおこなう権限を使います」という意味です。これを宣言せずにアプリの中でインターネットとの通信を実行しようとすると、「SecurityException」という例外が発生して通信に失敗します。

他にもいろんなパーミッションがありますが、詳しくは公式リファレンスを参照してください。

「AndroidManifest.xml」で宣言したパーミッションは、そのアプリをGoogle Playで公開したときにユーザーに提示されます（**図1**）。

図1：アプリをインストールする際のパーミッション表示

パーミッションのリファレンス
http://developer.android.com/reference/android/Manifest.permission.html

もしゲームのアプリが、ゲームで使う必要があるようには見えない権限（例:電話をかける権限）を宣言していたとしたら、ユーザーはそのアプリを信用してダウンロードしてくれるでしょうか？　おそらく信用されないでしょう。よって、アプリ開発者は、自分のアプリが使っている機能が必要としている権限だけを宣言しなければなりません。また、ユーザーが疑問に思わないように、どの権限は何のために使っているのかを、アプリの説明文で明らかにしておく方が、ユーザーの信頼を得やすいでしょう。

実装におけるセキュリティ

実装におけるセキュリティの問題は、主にアプリ開発時の不注意や、設計の不備によって引き起こされます。大きく以下のように分類されます。

- **アクセス制限の不備**
- **暗号化／暗号通信の不備**
- **その他**

「IPA」（独立行政法人情報処理推進機構）によると、Androidアプリで報告されたセキュリティ問題のうち、約75%がアクセス制限の不備でした。アクセス制限の不備は、さらにファイルのアクセス制限の不備と、コンポーネントのアクセス制限の不備に大別することができます。

以下では、まずファイルのアクセス制限について説明し、その後でコンポーネントのアクセス制限について説明します。

21-1-2　Android のファイルアクセス制限機能（初歩）

Androidアプリ開発では、フレームワークが用意している様々なアクセス制限の仕組みを利用することができます。ここでは、その中でもファイルへのアクセス制限について説明します。

アプリ開発をしていると、アプリの終了や端末の再起動をおこなっても、次回のアプリ利用時にはデータを覚えておいて欲しいことがあります。たとえば、設定情報、写真などのメディアデータ、編集途中のテキストメモなどが考えられます。そのために、データを保存しておくためのAPIが用意されていますが、間違った使い方をすると、そのデータはユーザーや他のアプリから簡単に参照されてしまう状態になります。

写真のデータであれば、他のアプリやユーザーからいつでも見られる状態になっている方が良いですが、サービスのアカウント情報（例:LINEのログイン情報）や、日記アプリに書き込んだテキストが、他のアプリから簡単に読めるようになっていてはいけません。

SharedPreferences

Androidアプリは、アプリから設定情報などのデータを簡単に保存しておく仕組みとして「SharedPreferences」という機能を提供しています。SharedPreferencesは、「キー」と「バリュー」を紐付けて記憶しておく簡易データベース（キーバリューストア）であり、以下のように使います。

リスト2

```
// context を取得する（例：Activity自身を入れる）
Context context = (Context)this;

// user_settings という名前のキーバリューストアを取得する
SharedPreferences prefs = context.getSharedPreferences("user_settings", Context.MODE_PRIVATE);

// 取得したキーバリューストアからユーザー名とパスワードを取得する
String username = prefs.getString("username");
String password = prefs.getString("password");
```

SharedPreferencesは、ActivityやServiceの親クラスである「Context」に定義されたメソッド、「getSharedPreference()」で取得することができます。親クラスで定義されているため、ActivityやServiceから利用しやすいものです。

Context#getSharedPreferences()の第1引数はキーバリューストアの名前です。アプリ内に複数のキーバリューストアを持つことができます。第2引数で、このキーバリューストアへのアクセス制限を指定します。

アクセス制限名	意 味
MODE_PRIVATE	このアプリからのみ読み書き可能
MODE_WORLD_READABLE	他のアプリからも読み出せる
MODE_WORLD_WRITEABLE	他のアプリからも書き込める

表1：第2引数に指定するアクセス制限

アプリを開発していると、自分が作成した複数のアプリ間で設定データを共用したいと考え、他のアプリからも読み書きできるようにしたくなる場合があるかもしれません。しかしそのときに、「MODE_WORLD_READABLE」や「MODE_WORLD_WRITEABLE」を指定して設定を作成すると、他人の作ったアプリからもその設定情報を勝手に読み書きできるようになってしまいます。そのため、機能としては用意されていますが、SharedPreferencesは「MODE_PRIVATE」以外では作成しないようにしましょう。

ちなみに、MODE_WORLD_READABLEおよびMODE_WORLD_WRITEABLEはAPI Level 17から「deprecated（将来バージョンのどこかで完全削除予定）」となったので、いつ新しいAndroidバージョンで使用できなくなってもおかしくありません。その点からも、MODE_WORLD_READABLE、MODE_

WORLD_WRITEABLEを使用することは勧められません。

では、複数のアプリ間でデータを安全に共有するにはどうしたらよいでしょうか? その仕組みの作り方については、後で説明します。

先ほど、保存済みのデータを読み出す例については説明しましたが、逆にデータを保存しておく方法は以下の通りです。

リスト3

```
String username;
String password;

// username と password にユーザー名とパスワードを格納しておく（省略）
......

// prefs の変更を受け付けるEditorオブジェクトを取得する
SharedPreferences.Editor editor = prefs.edit();

// 取得したEditorオブジェクトにキーとバリューの組み合わせを登録する
editor.putString("username", username);
editor.putString("password", password);

// 変更を確定する（非同期版）
editor.apply();
```

アプリ専用データ領域内のファイル

端末にアプリがインストールされたとき、「/data」ディレクトリー以下に、そのアプリ専用のフォルダーが作成されます。シングルユーザー利用の端末であれば、「/data/data/<アプリのパッケージ名>」というフォルダー名となります。

アプリが保存しておきたいデータ（ファイル）は、そのディレクトリー内に作成することができます。しかし、アプリのパッケージ名を取得して、このフォルダー名を自分で作ってアクセスするということをしてはいけません。Android 4.2からはタブレット端末で、Android 5.0からは電話型端末でも、マルチユーザーを利用できるようになり、必ずしも前述のフォルダー名が現在実行中のユーザー用のフォルダー名とはならないためです。

マルチユーザーでも正常に動作するようにするため、アプリ専用フォルダーへのアクセスは、「Context#openFileInput()」や「Context#openFileOutput()」などのAPIを使用してください。

Context#openFileInput()は、アプリ専用フォルダー内にあるファイルを指定して読み込み用に開くAPIで、Context#openFileOputput()は、アプリ専用フォルダー内にあるファイルを書きこみ用に開くAPIです。以下は、使い方の例です。

リスト4

```
// アプリ専用のフォルダーにファイルを書きこみ用に開く
OutputStream os = null;
try {
    os = context.openFileOutput("sample.txt", Context.MODE_PRIVATE);

    // os にファイルを書き出す(省略)
     ......
} finally {
    // ファイルを閉じる
    if (os != null) {
        os.close();
    }
}
```

リスト5

```
// アプリ専用フォルダー内のファイルを読み込み用に開く
InputStream is = null;
try {
    is = context.openFileInput("sample.txt");

    // isからファイルを読み出す(省略)
    ......
} finally {
    if (is != null) {
        is.close();
    }
}
```

勘の良い方は気づいたかもしれませんが、SharedPreferencesで指定したContext.MODE_PRIVATEは、このアプリ専用データ領域内でのファイル作成時のアクセス制限と同じものです。実は、SharedPreferencesのキーバリューを保存するファイルは、アプリ専用データ領域内に作成されます。SharedPreferencesの場合と同様、他のアプリからアクセスできなくするために、ファイル作成の場合でも同じように、Context.MODE_PRIVATEのみを使用してください。

拡張ストレージ

拡張ストレージは、特定のアプリからだけでなく、他のアプリや、端末を接続したPCからもアクセス出来る領域です。カメラアプリで撮影した写真、ブラウザーでダウンロードしたファイルなど、大きなファイルの置き場として主に利用されます。

セキュリティの観点から、以下のような注意が必要です。

- **他のアプリからのアクセスを止める方法がないため、センシティブな情報（個人情報や著作物）を生データでは保管しない**
- **著作物は暗号化して保管する**
- **他のアプリによってファイルを削除される可能性があるため、ファイルがなくなった場合の回復処理を考慮しておく**
  →例：購入した著作物を再ダウンロードできるようにする

上記のような制限について対策するのは大変なので、アプリ専用データ領域のみを使用するようにアプリを作っていればよいのではないかと考えるかもしれません。端末によって異なりますが、アプリ専用データ領域は小さく、拡張ストレージは大きいことが一般的です。よって、動画などの大きなデータを取り扱う場合には、制限事項を理解した上で、拡張ストレージを利用することを考えてください。

Android 4.4以降のSDカードアクセス

Android端末の中には、端末に内蔵した拡張ストレージとは別にSDカードを挿入できるものがあります。その場合、SDカードは2番目の拡張ストレージとして扱われます。Android 4.3までは、すべての拡張ストレージのアクセス制限は同じでしたが、Android 4.4以降から1番目と2番目以降の拡張ストレージでは、一般のアプリ権限でおこなえることが変わっています。

Android 4.4以降では、一般のアプリがたとえ「WRITE_EXTERNAL_STORAGE」のパーミッションを宣言して、ユーザーに許可されていたとしても、2番目以降の拡張ストレージにファイルを書き込むことができなくなりました。あなたが、カメラアプリや画像ギャラリーアプリを開発する場合、写真をSDカードにエクスポートするような機能を提供することはできないと考えてください。

システム権限を持つアプリは、SDカードへのファイルの書き込みが可能です。SDカードへのファイルコピーや移動ができるファイルマネージャ系のアプリは、メーカーがプリインストールで提供したもののみが完全な機能を提供できます。

演習問題

2つのアプリを作成します。1つ目のアプリでデータを作成し、2つ目のアプリはそのデータにアクセスを試みます。1つ目のアプリでデータ作成時に指定したアクセス制限が、2つ目のアプリからのアクセスの際に正しく働くことを確認しましょう。

1つ目のアプリのプロジェクト作成

ここではSDKバージョン指定を4.4(KitKat)にそろえてプロジェクトを作成します(**図2**)。新規Activityの作成画面では「Blank Activity」を選択してください。

図2:プロジェクトの新規作成

アプリ名、プロジェクト名はExportApp、パッケージ名はjp.techinstitute.exportsample としています。2つ目のアプリから参照する際にこのパッケージ名を使用しますので、他のパッケージ名にした場合は適宜読み替えてください。

次にデータ書き込み用のボタンとデータ読み込み用のボタンを持つ単純なレイアウトを作成します。

リスト6

```
<RelativeLayout
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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="jp.techinstitute.exportsample.MainActivity" >

    <Button
        android:id="@+id/write_button"
        android:text="Write Data"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true" />

    <Button
        android:id="@+id/read_button"
        android:text="Read Data"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_centerHorizontal="true"
        android:layout_below="@id/write_button" />

</RelativeLayout>
```

次にActivityのJavaコードを書いていきます。

リスト7

```
ackage jp.techinstitute.exportsample;

import android.app.Activity;
（以下import文省略）

public class MainActivity extends Activity
implements View.OnClickListener {

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

        Button writeButton = (Button) findViewById(R.id.write_button);
        Button readButton = (Button) findViewById(R.id.read_button);
        writeButton.setOnClickListener(this);
        readButton.setOnClickListener(this);
    }

    （以下、変更のないコード省略）

    @Override
    public void onClick(View v) {
        SharedPreferences prefs = getSharedPreferences("sample1",
MODE_PRIVATE);

        switch(v.getId()) {
        case R.id.write_button:
            SharedPreferences.Editor editor = prefs.edit();
            editor.putString("sample_string", "Data is stored!!");
            editor.apply();
            break;
        case R.id.read_button:
            String readData = prefs.getString("sample_string", null);

            if (readData == null) {
                readData = "Data isn't stored.";
            }

            Toast.makeText(this, readData, Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

パッケージのインポートエラーが出る部分は、Eclipseの補完機能を利用して適宜解消してください。

四角で囲った部分が今回特有のコードです。読み込みボタンを押したときには、データが書き込まれていればそのデータを、書き込まれていなければ書き込まれていないことをトーストで表示します。

上記ではデータの書き込みは、Context.MODE_PRIVATEで行っています。

2つ目のアプリのプロジェクト作成

2つ目のアプリプロジェクトを1つ目のプロジェクトと同様に作成します。アプリ名、プロジェクト名をUseApp、パッケージ名を jp.techinstitute.usesample としてください。

次にデータ読み込み用のボタンを持つ単純なレイアウトを作成します。

リスト8

```
<RelativeLayout
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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="jp.techinstitute.exportsample.MainActivity" >

    <Button
        android:id="@+id/read_button"
        android:text="Read Other App Data"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true" />

</RelativeLayout>
```

次にActivityのJavaコードを書いていきます。

リスト9

```
package jp.techinstitute.usesample;

import android.app.Activity;
(以下import文省略)

public class MainActivity extends Activity
implements View.OnClickListener {

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

        Button button = (Button) findViewById(R.id.read_button);
        button.setOnClickListener(this);
    }

(以下、変更のないコード省略)

   @Override
    public void onClick(View v) {
        Context context;
        try {
            context =
createPackageContext("jp.techinstitute.exportsample",
                                          CONTEXT_RESTRICTED);
            SharedPreferences prefs =
context.getSharedPreferences("sample1",
MODE_PRIVATE);

            String readData = prefs.getString("sample_string", null);

            if (readData == null) {
                readData = "data isn't stored.";
            }

            Toast.makeText(this, readData, Toast.LENGTH_LONG).show();
        } catch (NameNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

ここでは、jp.techinstitute.exportsampleのコンテキストを取得して、それが持つSharedPreferences を読み取ろうとしています。しかし、MODE_PRIVATEで作成されているため、データを読み取れないメッセージがトーストで表示されます。

それでは、わざと制限を弱くして、別のアプリから読み取れるようにしてみましょう。

すでにsample1の名称でMODE_PRIVATEのSharedPreferencesは作成済みであるため、sample2という名称でMODE_WORLD_READABLEなものを作ります。jp.techinstitute.exportsample の変更点は以下の通りです。

リスト10

```
@Override
    public void onClick(View v) {
        SharedPreferences prefs = getSharedPreferences("samp
le2",
MODE_WORLD_READABLE);

    （以下省略）
```

deprecatedなため、取り消し線が表示されますが無視してください。

同様にjp.techinstitute.usesample も変更します。

リスト11

```
@Override
    public void onClick(View v) {
       （省略）
       SharedPreferences prefs = context.getSharedPreferences("
sample2",
 MODE_PRIVATE);
MODE_PRIVATE);

    （以下省略）
```

MODE_PRIVATEはそのままで、sample1をsample2に変更するだけです。

変更することで、このように非常に簡単に別のアプリからデータを読み出せることが確認できました。これは、他のアプリから情報を簡単に盗むことができるということです。

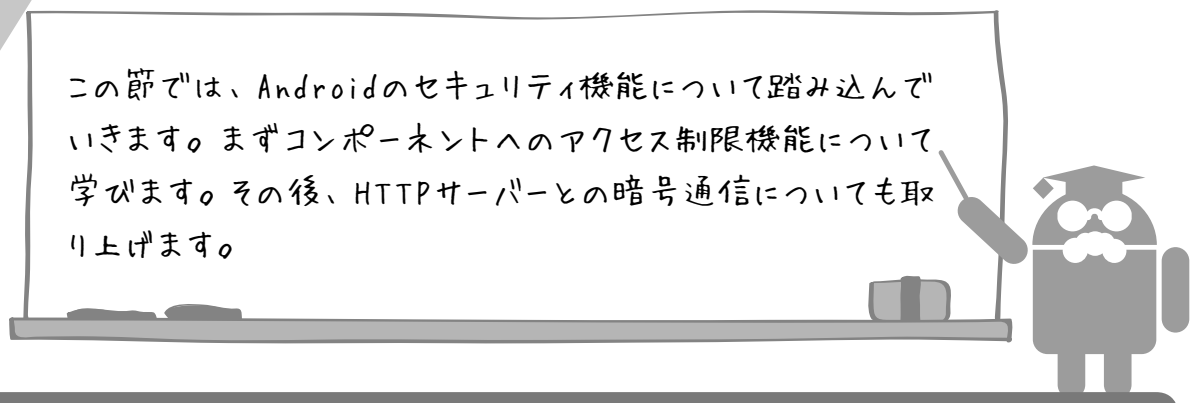
別のアプリから読み取れる設定で、パスワードのような重要な情報を保存しておくことなどは、危険なので絶対にしないようにしましょう。

まとめ

アプリのセキュリティについて、仕様と実装の両面で考慮しないといけないことを本節で学びました。また、具体的にどのようにアクセス制限が働くかを確認しました。

21-2 Androidアプリの高度なセキュリティ

著：中西良明



この節で学ぶこと

- Androidアプリのコンポーネントの守り方
- HTTPサーバーとの暗号通信

この節で出てくるキーワード一覧

android:exported
permission
grantUriPermission
protectionLevel
SSL/TLS

この章で出てくるクラス

ContentProvider
URL
HttpsURLConnection
InputStream
InputStreamReader
BufferedReader

21-2-1 コンポーネントのアクセス制限

攻撃があったことは確認されていませんが、有名なTwitterクライアントアプリが実際に持っていた脆弱性です。

Androidのフレームワークが用意している様々なアクセス制限のうち、ファイルへのアクセス制限については前節で説明しました。本節ではコンポーネントへのアクセス制限を説明します。Androidでは、ファイルレベルのアクセス制限だけではなく、アプリの構成要素（コンポーネント）に対してもアクセスを制限することができます。

Androidアプリを開発していると、複数のActivityやServiceなどを作成していくことがよくあります。それらのコンポーネントの中には、外部アプリから自由に呼び出せてはいけない機能を持ったものがあるかもしれません。たとえば、SNSクライアントアプリの中のあるコンポーネントを、悪意のある他のアプリから呼び出すことができるようになっていると、SNSクライアントアプリの権限でネットに公開するつもりのなかった端末内の写真を全世界にばら撒かれるということが起こりえます。

Androidにはこれを防ぐ仕組みとして、AndroidManifest.xmlにてActivityやServiceなどに「android:exported」属性を宣言しておくことができます。以下はその記述例です。

リスト1

```
<application ...>
    <activity android:name="com.example.android.activity1"
        android:exported="true">
        ......
    </activity>

    ......

    <servce android:name="com.example.android.service1"
        andorid:exported="false">
        ......
    </service>

    ......
</application>
```

android:exportedを「false」にしておくことで、そのActivity、Serviceなどを他のアプリに対して非公開状態にすることができます。その場合も、自分のアプリ内からは呼び出すことが可能です。セキュリティの観点からは、公開にする必要がないコンポーネントにはandorid:exportedをfalseに設定することを習慣づけましょう。

特定のアプリにだけデータを開放する（grantUriPermission）

Androidでは、アプリ内およびアプリ間でのデータのやり取りに便利な仕組みとして、「ContentProvider」というものを持っています。ContentProviderの詳細は本節では省略しますが、ContentProviderを介することで、アプリは管理するデータを他のアプリからアクセスできるようにすることができます。

ただし、アプリ内でContext.MODE_PRIVATEで保管し、他のアプリからのアクセスを防いでいたにもかかわらず、ContentProviderを通じて他のアプリすべてにアクセスを開放してしまうことは望ましくないでしょう。特定のアプリにだけ、自分が作ったContentProviderにアクセスさせるために、「grantUriPermission」を使うことができます。

まず、AndoridManifest.xml でContentProvidrerの宣言にgrantUriPermissionを使用することを宣言します。

リスト2

```
<provider andorid:name=".SampleProvider"
         android:authorities=" com.example.android.sampleprovider"
         android:grantUriPermission="true">
</provider>
```

そして、自分が作ったContentProviderの中で、どのアプリからのアクセスを許可するかを宣言します。

リスト3

```
public class SampleProvider extends ContentProvider {
    ...
    @Override
    public void onCreate() {
        ......
        // パッケージ名 com.example.android.reader のアプリにアクセスを許可する
        grantUriPermission("com.example.android.reader",
             Uri.parse("content://com.example.sampleprovider/",
             Intent.FLAG_GRANT_READ_URI_PERMISSION);
        ......
    }

    ......
}
```

「grantUriPermission()」の第1引数で許可を与えるアプリのパッケージ名を指定し、第2引数ではアクセスを許可するContentProviderのURI、第3引数で許可を与えるアクセスの種類（ここでは読み込みアクセス）を指定します。上記の例では、「com.example.android.sampleprovider」のContentProviderは、

「com.example.android.reader」からの読み込みアクセスを受け入れ、他のアプリからのアクセスを拒絶するようになります。

grantUriPermission()メソッドにて与えた許可は、端末の再起動まで与えられたままとなります。権限を取り上げたい場合は、「revokeUriPermission()」メソッドを呼び出すことができます。

リスト4

```
revokeUriPermission(Uri.parse("content://com.example.sampleprovider/"),
        Intent.FLAG_GRANT_READ_URI_PERMISSION);
```

revokeUriPermission()では、許可を取り上げるアプリのパッケージ名指定ができないため、指定したURIのContentProviderへのアクセスをすべて取り除くことに注意してください。必要であれば、すべて取り除いた後に、許可するアプリだけを再度登録しなおしてください。

同じ鍵で署名したアプリのアクセスを許可する

grantUriPermissionでは、特定のアプリを明示的に指定して、ContentProviderにアクセスを許可することができました。これは、事前に許可する対象のアプリが明確な場合には有効です。

では、自分が開発したコンポーネントを、将来自分が作成するアプリから自由に利用できるようにしつつ、他人のアプリからのアクセスを禁止するのはどうすればよいのでしょうか？　また、ActivityやServiceに対しても、自作アプリからのアクセスのみを許可したい場合はどうすればよいでしょうか?

Androidにはそのための仕組みが用意されています。それがパーミッションです。

(1)コンポーネントを提供するアプリのAndroidManifest.xmlでパーミッションの作成をおこなう。また、同じ鍵で署名したアプリのみアクセスを許可する旨を記載する

(2)上記アプリと同じ鍵で署名した別のアプリのAndroidManifest.xmlで、(1)で作成したパーミッションの利用を宣言する

実はパーミッションは、すでに述べたシステムが用意したものだけではなく、アプリが独自に追加することも可能となっています。そしてそれを利用することで、自分が開発したアプリにのみ権限を与えることが可能となります。

まず、(1)のアプリのAndroidManifest.xmlの例を見てみましょう。

リスト5

```
<permission andorid:name="com.example.android.CUSTOM_PERMISSION"
            android:description="Custom Permission"
            android:label="Custom Permission"
            android:protectionLevel="signature" />

<uses-permission android:name="com.example.android.CUSTOM_PERMISSION" />

<application ...>
    <activity android:permission="com.example.android.CUSTOM_PERMISSION"
              .... >
        ......
    </activity>
    ......
</application>
```

まず「<permission>」タグで新しいパーミッションを登録します。名前は自由につけることが可能です。「android:protectionLevel」に「signature」を設定していますが、これが同じ鍵で署名したアプリにだけ許可を与えるということを示します。

permissionに設定する項目の詳細は、以下の公式リファレンスを参照してください。

http://developer.android.com/guide/topics/manifest/permission-element.html

次に、自分が宣言した<permission>を自分で「<uses-permission>」で宣言しておくことで、自分自身もそのパーミッションを利用できるようにしています。Activityの宣言で「android:permission」に宣言したパーミッション名を記述することで、このActivityは同じ鍵で署名したアプリからしか呼び出せなくなります。

呼び出す側のアプリは簡単で、インターネットアクセスのパーミッションを要求した場合と同じように、以下の記述をAndroidManifest.xmlに記載すればよいのです。

リスト6

```
<uses-permission android:name="com.example.android.CUSTOM_PERMISSION" />
```

上の記述を同じ鍵で署名していないアプリが宣言していても、許可は与えられません。

この仕組みは非常に簡単で、かつ十分なセキュリティを備えているように見えますが、1つ大きな弱点があります。<permission>でのパーミッションの登録は、先にインストールしたアプリのものが優先されることです。パーミッション名を、自分が管理するドメイン名を用いたものにするなど、工夫することで意図しない衝突は避けることが

できます。しかし、悪意のある開発者によるアプリが、事前に同じ名前のパーミッションを弱い制限（誰でも使える制限）で登録していた場合、後からインストールされた正しいアプリによる制限は働きません。

このように、独自パーミッションによる制限は正しく働かない場合があるので、独自パーミッションをあてにした設計を行わないことを推奨します。

ユーザーの識別には端末ID（UDID）を使ってはいけない

アプリを開発していると、アプリを利用しいているユーザーを識別したくなる場合があります（例：ゲームのスコア管理、SNSの簡易ログイン）。

その方法として、端末固有のID（Unique Device Identifier）を利用するアプリがありますが、それにはセキュリティ上の問題があります。たとえばSNSの簡易ログインにUDIDを使用した場合、以下のような攻撃方法が考えられます

(1) 悪意のあるアプリがUDIDを取得する
(2) 取得したUDIDを用いてSNSのサーバーにログインする
(3) ユーザーの情報をSNSのサーバーから取得する

UDIDはハードウェアと紐づいていて変更することもできないため、それを悪意のあるアプリが取得してしまえば、攻撃を防ぐことができなくなります。

アプリで簡易ログインのためになんらかのIDを使う場合は、UDIDではなくUUID（Universally Unique Identifier）を利用してください。Androidでは、以下の方法でUUIDを生成することが可能です。

```
String uuid = UUID.randomUUID().toString();
```

そして、生成したUUIDはSharedPreferencesに保存しておくことで、次回以降の簡易ログインなどに使用してください。

ただ、真の意味ではこれはユニーク（唯一の）IDとはなっていません。ランダム生成されたIDなので、実際には、他のIDと重複していないかをサーバーとの通信で確認し、重複がないことを確認するまで生成し直すなどして重複を排除しなければなりません。サーバーでユニークIDを生成して、端末アプリに対して与えるという設計とするのもよいでしょう。

21-2-2 通信のセキュリティ

ユーザーにとって魅力的なアプリを作ろうとすれば、アプリはサーバーとの通信をおこなうことが多くなります。たとえばソーシャルゲームはサーバーと連携して、ユーザーに対戦や協力プレイの機能を提供します。ニュースなどのキュレーション（情報収集とまとめ）をおこなうアプリも、サーバーと連携して、キュレーションした結果を取得するなどが必要となります。

特にログイン時の通信は傍受されると、悪意のあるユーザーやアプリによってアカウントの乗っ取りなどがおこなわれてしまう危険性があります。よってそのような通信については、アプリとサーバーとの間の通信は暗号化することが必要です。

ここでは、Androidアプリがサーバーと安全な通信路を確立するための方法について説明します。

SSL/TLSによる暗号通信

厳密にはTCP/IP以外でも使用できるものです。

「SSL」（Secure Socket Layer）および「TLS」（Transport Layer Security）は、TCP/IP上で安全な通信路を構築するためのプロトコルです。

それらの通信の安全性についての説明、および公開鍵暗号方式の説明は、本教科書の趣旨から外れますので詳細を割愛しますが、以下のような手順でクライアントとサーバーの間の安全な通信路を確立します。

(1)クライアントがサーバーに接続を要求する
(2)サーバーが応答し、クライアントにサーバーの証明書を送る
(3)クライアントはサーバー証明書がCA(Certificate Authority(認証局))から発行された正しいものかどうかを検証する。正しい場合、クライアントで生成した乱数をサーバー証明書に含まれる公開鍵で暗号化してサーバーに返す
(4)サーバーは、暗号化された乱数データをサーバーの秘密鍵で復号(暗号解読)する
(5)サーバーとクライアントは、共有した乱数データから共通鍵暗号の鍵を生成し、以降の通信をその共通鍵を用いて暗号化しながらおこなう

さらにクライアントが証明書を送付し、サーバーがクライアントを検証する場合もあります。

アプリからサーバーと通信する場合、ほとんどの場合は「HTTP」を用いることとなります。SSL/TLSによる安全な通信路でおこなうHTTPを、特に「HTTPS」と呼びます。サーバーとの接続でHTTPSが利用される場合、URLの先頭は「https://」から始まるものとなります。

Androidでは、HTTPSによるサーバーとの接続を簡単におこなうことができます。実はHTTPの通信の場合とほとんど手順は同じで、「URL」クラスと「HttpsUrlConnection」を用います。以下に手順を示します。

リスト7

```
try {
     // URLクラスのインスタンスを作成
     URL url = new URL("https://www.example.com/");

     // サーバー接続用のオブジェクトを生成する
     HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();

     // リクエストメソッドなどのオプション設定
     conn.setRequestMethod("GET");
     conn.setInstanceFollowRedirects(false);
     conn.setRequestProperty("Accept-Language", "jp");

     // サーバーと接続する
     conn.connect();
} catch (IOException e) {
    // エラー処理を記述する
}
```

サーバーとの接続が失敗すると「IOException」が発生しますが、接続に失敗する理由は1つではありません。以下のようなものが考えられます。

・サーバーが見つからない
・サーバーとのTCP/IPの接続に失敗した
・サーバーの証明書が正しくない

特に3番目がセキュリティでは重要です。このエラーが出た場合、アプリとサーバーとの間に割り込んで、通信を傍受しようとしている何者かがいるのかもしれません。その場合は単にサーバーへの接続が失敗したというエラーを出すのではなく、偽のサーバーに接続しようとしているかもしれないことを警告するエラーをユーザーに表示し、注意を喚起することが望ましいです。

そのようなセキュリティ警告を意識したサーバーへの接続処理の書き方を、Googleのサーバーに HTTPSで接続する演習を通じて学びます。

アプリのプロジェクト作成

アプリプロジェクトを作成します。作成方法は前節の場合と同様で、API LevelをAndroid4.4(KitKat)、アプリ名、プロジェクト名をTLSAppパッケージ名の部分は jp.techinstitute.tlssample としてください。

次にWebViewを持つ単純なレイアウトを作成します。

リスト8

```
<RelativeLayout
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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="jp.techinstitute.tlssample.MainActivity" >

    <WebView
        android:id="@+id/webview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```

次に、ActivityのJavaコードを記述します。

サーバからのデータ読み込みを行い、取得した結果をWebViewに表示する処理となります。

リスト9

```
package jp.techinstitute.tlssample;

（import文は省略します）

public class MainActivity extends Activity {
    private static final String SITE_URL =
 "https://www.google.co.jp/";

    private WebView mWebView;

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

        mWebView = (WebView) findViewById(R.id.webview);

        AsyncTask<Void, Void, String> task = new AsyncTask<Void, Void, String>() {

            private String mSecurityReason;

            @Override
            protected String doInBackground(Void... params) {
                try {
                    URL url = new URL(SITE_URL);
                    HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();

                    conn.setRequestMethod("GET");
                    conn.setInstanceFollowRedirects(false);
                    conn.setRequestProperty("Accept-Language", "ja-jp");
                    conn.setRequestProperty("Accept-Charset", "utf-8");
                    conn.setRequestProperty("User-Agent", "Mozilla/5.0");

                    conn.connect();

                    InputStream in = conn.getInputStream();
                    BufferedReader br = new BufferedReader(new InputStreamReader(in, "UTF-8"));

                    StringBuilder sb = new StringBuilder();
                    String line;
                    while((line = br.readLine()) != null) {
                        sb.append(line);
                        sb.append("\n");
                    }

                    return sb.toString();
                } catch (SSLException e) {
                    e.printStackTrace();
                    mSecurityReason = e.toString();
                } catch (IOException e) {
                    e.printStackTrace();
```

```
}

                return null;
            }

            @Override
            protected void onPostExecute(String result) {
                if (result != null) {
                    mWebView.loadDataWithBaseURL(SITE_URL, result,
 "text/html", "UTF-8", null);
                } else if (mSecurityReason != null) {
                    Toast.makeText(MainActivity.this, mSecurityReason,
 Toast.LENGTH_LONG).show();
                }
            };

        };

        task.execute();
    }

（以下、省略します）
```

サーバーとの接続時にHttpsURLConnection#setProperty()で様々な指定を行っていますが、これは文字コードUTF-8で結果を返してもらうためのものです。ここでは詳細を割愛します。

Androidではメインスレッド（UIスレッド）で通信を行ってはいけないので、AsyncTaskを用いてバックグランドで通信し、結果をUIスレッド上で表示するようにしています。成功すればページが表示されます。

SSL/TLSに失敗する（例:証明書検証に失敗する）などの場合は、失敗時の例外（SSLException）を人間が読めるものにした情報をトーストで表示するようにしています。

上記のGoogleサーバーへの接続については、インターネット上に障害が発生するなどの何らかのトラブルが無ければ、通常は成功するはずです。

SSL/TLSの失敗の確認

では、SSL/TLSの接続が失敗する場合も試してみましょう。ただし、通信の間に割り込みエラーを発生させるのは難しいため、エラーを別の方法で発生させてみます。

サーバーによっては、クライアント側にも証明書を要求する場合があるということについて少しだけ触れましたが、そのような要求を行うところとして典型的なものに、銀

行のサイトがあります。他にもお役所のサイト(e-Taxなど)でクライアント側に証明書を要求する場合があります。

具体的なURLについてはここには記述しませんが、Google検索などで銀行のサイトを見つけ、その中から法人向けのネットバンキングサービスなどのログインを探し、クライアント証明書を求めるURLを見つけてみてください。

そして、先ほど指定したGoogleのサーバのURLをそのURLに置き換えたアプリを実行してみましょう。具体的には以下の定数定義を変更するだけです。

```
private static final String SITE_URL =
 "https://www.google.co.jp/";
```

この部分を見つけたURLに変更し、ビルドしてアプリをインストールし直して実行してみましょう。クライアント証明書を指定する処理が記述されていない、サーバーが接続を拒否し、例外(SSLException)が発生します。

通信に割り込んでデータを盗もうとする中間者攻撃では、この場合とはメッセージ内容などが異なる例外が発生しますが、基本的には同じです。

実際のアプリでは、SSL接続での例外発生は適切にユーザーにエラー内容を提示するようにしてください。

独自証明書を用いたサーバーとのHTTPS接続

アプリの開発をする際に、サーバーも同時に開発をするということがよくあります。ソーシャルゲームで、端末アプリとサーバーの開発をおこなう場合などもその一例です。

本番運用となれば、ほとんどの場合はサーバーには正式な証明書を設定することになりますが、開発中には様々な理由から正式な証明書を置くことが出来ない場合があります。

開発用のサーバーは本番用サーバーとURLが異なり、本番用の証明書が利用できない、まだ証明書を発行していない、などなど。

その場合、ひとまずはHTTPSではなくHTTPで開発を進めていくというのも1つの手ですが、できるだけ本番環境に近づけて、本番環境で起こる問題を減らすのは大事です。

また、社内システムと接続するアプリの場合、その会社独自のCAから発行された証明書が設定されたサーバーを、本番でも利用していることがあります。ただし、そのような独自証明書を持ったサーバーとのHTTPS接続では、Androidに標準で備わっているCA証明書を用いて、HTTPSサーバーの正当性を証明することができません。

ネット上には、証明書検証でのエラーを無視する処理を書くことで、エラーをとりあえず回避するという対策がよく出ています。一時的な対策としては良いのですが、セ

その他にも、ホスト名検証に失敗する場合の対策など、よく起こるケースについての対策が示されています。

キュリティ上はあまり良い方法であるとは言えません。特に、本番サーバーでも独自証明書を利用する場合には絶対におこなってはいけない対策です。

実は、独自証明書を用いたHTTPSサーバーとの接続時に、サーバーの正当性を検証する方法は、Android公式サイトにある以下のドキュメントに詳しく説明されています。

http://developer.android.com/training/articles/security-ssl.html

そちらに記載されているサンプルをベースに、使い方を日本語コメントで説明します。

リスト10

```
// まずCertificateFactoryのインスタンスを生成します
CertificateFactory cf = CertificateFactory.getInstance("X.509");

// 独自証明書を発行したCA証明書を読み込みます
// この例では、ワシントン大学ローカルのCA証明書です
// From https://www.washington.edu/itconnect/security/ca/load-der.crt
InputStream caInput = new BufferedInputStream(new FileInputStream("load-der.crt"));
Certificate ca;
try {
    ca = cf.generateCertificate(caInput);
    System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
} finally {
    caInput.close();
}

// 前述のCAを含んだKeyStoreを作成します
String keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);

// 前述のKeyStoreを持ったTrustManagerFactoryを作成します
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);

// SSLContextを前述のTrustManagerFactoryから取得した
// TrustManagerを用いて作成します
SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);

// HttpsUrlConnectionに、ここまでの手順で作成したSSLContextを設定します
URL url = new URL("https://certs.cac.washington.edu/CAtest/");
HttpsURLConnection urlConnection =
    (HttpsURLConnection)url.openConnection();
urlConnection.setSSLSocketFactory(context.getSocketFactory());

// urlConnectionからの入力を、コンソールに出力する
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out)
```

実はHTTPS接続においては、独自の証明書を持ったサーバーとの、接続時の証明書検証の不備をセキュリティ問題として指摘される事例は多く、大手の会社がリリースしたアプリでもそういった例がありました。

気づかれないだろうと思って、証明書検証を省略するなどの手抜きをした結果、ネット上でのその手抜きを大きく話題にされてしまうということになりかねません。ご注意ください。

https://jvn.jp/vu/JVNVU90369988/

まとめ

さまざまなセキュリティ対策を行うことは、機能を増やしてアプリを魅力的にすることにはつながらないため、後回しにされがちです。場合によっては、あえてその作業を減らして開発をスピードアップするといったことも行われることがあります。

しかし、セキュリティを軽視した開発をすると、そこを攻撃されたときに大きなダメージを受けてしまい、開発の時点でかかったコストの何倍、何十倍、何百倍ものコストを支払うことになるかもしれません。

現代のネットには、セキュリティ不備に対する攻撃があふれています。セキュリティ対策は転ばぬ先の杖ではなく、ネット上にサービス、アプリを提供する際に必須の防具です。その防具が無ければ殺されるのだというくらいの考えでいても、大げさではない時代です。有名になればなるほど、攻撃もまた増えます。

新しいアプリやサービスで世界に名を轟かせる野心を持つのであれば、攻撃されてもびくともしないセキュリティという防具も持ちましょう。