Tech Institute
第21章
セキュリティ

2015/06/07

講師: 猪俣 充央

自己紹介

- 株式会社 tech vein
  代表取締役　猪俣　充央 (twitter: @ino2222)
- 現在は主にiOSアプリ, Androidアプリを作る仕事をしています。
- 過去には、ケータイ向けのWebサービスやサーバ周りの開発もしていました。

今日のタイムスケジュール

- 13:00-15:00　セキュリティ前半
- 15:00-15:30　30分休憩
- 15:30-17:30　セキュリティ後半
- 17:30-17:40　10分休憩
- 17:40-19:00　サーバの復習

はじめに

- 今日の実習で必要なファイルを予めダウンロードしておいてください。

- http://bit.ly/1Mvx6Pj

セキュリティとは

- システムの不正利用や機密情報の漏洩、外部からの攻撃などを防ぐための技術的に保護して、危険を排除すること。

- システムに攻撃される危険がある状態、穴がある状態のことを、脆弱性(セキュリティ不備)がある、といいます。

- よくニュースになる「個人情報漏えい」「不正アクセス」はどこかのシステムに脆弱性があったために発生したものです。

  コンピュータ・サーバ・アプリなど、システムが絡むものにはすべて、脆弱性ができる可能性があります。

自分が作ったアプリに脆弱性があると…

- アプリを使っていたら個人情報が盗まれた！
  →アプリのユーザが減る。炎上する。
  最悪ユーザに訴えられてしまうケースも。
- ゲームで無限に魔法石を増やす裏ワザが見つかった！
  →課金されなくなる。ゲームバランスが崩れ、ゲームシステムが崩壊する。ずるをしないユーザが離れていく

脆弱性があると、せっかく作ったアプリが使われないどころか、大きな損害が発生することもある。
脆弱性は作らない、残さない事がとても大事！

仕様にセキュリティ不備があった事例
浮気防止アプリ(1)

- アプリをインストールした端末でのユーザの行動を監視する機能を持ったアプリ
- 位置情報、誰に電話したか、誰にメールしたか、などなど
- 取得したデータは、サービスを利用しているユーザ(彼女、妻など)に対して送られる
- ターゲットユーザは、彼氏(彼女)を束縛したいパートナー
- 監視対象(彼氏、夫など)の同意を得てインストールすることを前提としていた

仕様にセキュリティ不備があった事例
浮気防止アプリ(2)

- 問題視された点
  - 象が同意する前提はただの建前として観られ、実質的には同意なしでインストールされる不正アプリとみなされた

- 結果どうなったか？
  - サービス中止になった。
  - 名前と機能を若干変更したアプリをリリースしたが、それもすでに閉鎖されている。

仕様にセキュリティ不備があった事例
SNSアプリによる電話帳収集(1)

- アプリのユーザ拡大のため、アプリをインストールした端末から電話帳の情報を収集し、サーバに送ってデータベースを構築していた。
- SNS運営会社は、収集したメールアドレスに対して、勧誘のメールを送信するなどの宣伝活動を実施した。

仕様にセキュリティ不備があった事例
SNSアプリによる電話帳収集(2)

- 問題視されたこと
  - ユーザが意図しないところで勝手に電話帳が抜き取られた。

  - 電話帳はユーザの持ち物だが、中の個人情報はその人のものであること。

- 結果どうなったか?
  - ユーザが許可しないかぎり、電話帳情報を収集しない仕様に変更した。

対象ユーザが変わると
問題ではなくなる場合もある

- ターゲット:小学生を持つ親
- 要求　:子どもを見守りたい

- → 子どもの電話帳データを確認する

仕様は浮気防止アプリに似ていても、上記のようなサービスなら社会的にはOKとみなされた。

- 参考:イオンがスマートフォンの親子セットを発売開始
- http://itpro.nikkeibp.co.jp/atcl/news/14/111201862/

仕様のセキュリティ不備は影響が非常に大きい

- 仕様を客観的に見て、誰かに損害がないか、社会的に問題がないかを確認することが大事。
- 問題があった場合、アプリの前提を崩すので、非常に対応コストがかかったり、取り返しにつかないことになる場合もある。

実装にセキュリティ不備があった事例

- アプリがユーザのID/パスワードのような機微情報をSDカード上に暗号化せずに保存していた
- アプリ内で管理しているデータに、外部からアクセスできる状態になっていた
- サーバとの暗号通信で、弱い暗号方式を使用していた
- 開発用のログが本番アプリに残っていたため、ユーザの個人情報が漏れていた

- →これらの問題は特定のアプリに限ったものではなく、新しいアプリで何度も繰り返されています。

アプリのセキュリティ不備の傾向

- IPAによると、問題の75%はアクセス制限の不備

- https://www.ipa.go.jp/about/technicalwatch/20120613.html

- 『Android 特有の設定内容が開発者に周知できておらず、結果的に、アクセス制限の不備の脆弱性 を作り込んでしまっているのではないかと IPA では推測する。』

アクセス制御

アクセス制限

- ファイルなどの保存データを読み書きを誰に許可するかの制御のこと。
- アプリ上のデータはローカルファイル(端末内のファイル)やサーバに保存しないと、消えてしまう。
- 消えないように保存しないといけないけど、保存したものが外部に見えたり勝手に書き換えられたりすると困る場合が多い。
  - 例:アドレス帳の連絡先データ、LINEの会話の履歴、ゲームのセーブデータなど。

Androidのデータ保存先

- アプリ専用データフォルダ
  - アプリ内だけで使う画像、データなど任意のファイル保存用の領域
- 拡張ストレージ
  - 内部ストレージ・SDカードなど。
  - アプリ外と共有したい画像・ファイル、または外から参照されてもいいファイルの保存用の領域
- SharedPreferences
  - (実体はアプリ専用データフォルダ内XMLファイル)
- ローカルデータベース
  - (実体はアプリ専用データフォルダ内SQLiteファイル)

SharedPreferences

- アプリ内**だけ**で使う設定などの少量のテキストデータ保存用のクラス。
- 簡単に読み書きできるので、ちょっとした保存先として便利です。

- ただし実体は、アプリ専用領域に保存されるXMLファイルなので、あまり大きな(数百〜千件とかの)データの取り扱いは苦手です。

- **ゲームアプリでの利用例**
  - キャラクターの名前や効果音のON/OFFなど、ユーザごとの独自設定
  - ベストスコア、ゲームの進行状況、プレイヤーのステータスなどの状態保存

SharedPreferences の使い方

```
public class MyActivity extends Activity {

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

        SharedPreferences pref = getSharedPreferences(“pref",
                MODE_PRIVATE);
        SharedPreferences.Editor editor = pref.edit();
        // プレイヤー名
        editor.putString("name", "山田太郎");
        // 年齢
        editor.putLong("age", 19);
        // ベストスコア
        editor.putInt("bestScore", 100);
        // 効果音再生: オン
        editor.putBoolean("sound", true);

        editor.apply(); // 保存
    }
}
```

MyActivity.java

書き込み

SharedPreferences#edit()

Editor#putXXX()

Editor#apply()

pref.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="name">山田太郎</string>
  <int name="age" value="19" />
  <long name=“bestScore" value="100" />
  <boolean name="sound" value="true" />
</map>
```

読み込み

SharedPreferences#getXXX()

SharedPreferencesの
アクセス制御(MODE_WORLD_〜)

- MODE_WORLD_READABLE/MODE_WORLD_WRITABLE を使えば他のアプリからも読み書きができるようになります。
- 自分の作ったアプリ間でデータを共有したいときなどに使えるので一見便利そうですが、使ってはいけません。
- 理由は外部アプリごとの権限の制御ができず、誰でも(悪いアプリからも！)読み書きが出来てしまう問題があるためです。
- Deprecatedとなっており、今はまだ使えますがいずれ削除される機能です。

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

SharedPreferencesの
間違った使い方

- 開発したアプリAで保存したIDとパスワードを、別のアプリBと共用すれば便利になるぞ!

- アプリAで保存するSharedPreferences を、MODE_WORLD_READABLEに設定しよう!

- ということは絶対にやめてください。
- MODE_WORLD_READABLE なら、他人が作ったアプリからも読み出せてしまいます。

アプリ間の安全なデータ共有方法

- じゃあどうするか？

- もし自分で作ったアプリ間(＝信頼できるアプリ同士)だけでデータ共有したくなったときは ContentProvider を使いましょう。

- ContentProviderはアプリ内やアプリ間でデータをやりとりするのに便利な仕組みです。(教科書60〜64ページ)

- あとでまた少し出てきます。

専用データフォルダ

- アプリはインストール時に専用のファイル保存領域(フォルダ)を与えられます。
- アプリ内で使用する任意のファイルの保存先として使用できます。

- **実際の利用例**
  - サーバからダウンロードしたデータファイルの保存
    - 画像、csvファイル、データベースファイル etc.

専用データフォルダのファイルアクセス

- 専用データフォルダにファイルを保存したり開いたりするには、以下のメソッドを使います。
  - Context#openFileInput() // 読み込み用
  - Context#openFileOutput() // 書き込み用
- 専用データフォルダの場所はOSバージョンや環境によって異なります。

モード	専用データフォルダの場所
シングルユーザ	/data/data/<アプリパッケージ名>/
マルチユーザ(※)	/data/data/<ユーザID>/<アプリパッケージ名>/

※マルチユーザに対応しているのはAndroid4.2以降のタブレット、Android5.0以降のスマートフォンです。

ファイルの読み出し例

```
// アプリ専用フォルダ内のファイルを読み込み用に開く
InputStream is = null;
try {
    is = openFileInput("sample.txt");
    // is(入力ストリーム)からファイルを読み出す(省略)
    .....
} finally {
    if (is != null) {
        is.close(); // ファイルを閉じる
    }
}
```

ファイルの書き込み例

```
// アプリ専用フォルダにファイルを作って書きこみ用に開く
OutputStream os = null;
try {
   os = openFileOutput("sample.txt", MODE_PRIVATE);
   // os(出力ストリーム)にファイルを書き出す(省略)
      ......
} finally {
   if (os != null) {
         os.close(); // ファイルを閉じる
   }
}
```

専用データフォルダのファイルアクセス権限

- SharedPreferencesと同じ方法でアクセス制限名を指定します。
- SharedPreferencesは実は専用データフォルダ内にXMLファイルとして保存されています。
- SharedPreferencesと同様、MODE_PRIVATEだけを使いましょう。

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

その他の専用データフォルダ向け ファイルアクセスAPI

- Context#getFilesDir()
  - ファイルを置くディレクトリパスを取得する
  - openInputFile()/openOutputFile()のファイルはここに置かれる
- Context#getCacheDir()
  - キャッシュファイルを置くディレクトリパスを取得する
  - 一時的に使うファイル(キャッシュデータ)などはこちらに作ることが推奨されます

拡張ストレージ

- どのアプリからでも読み書きできる共有領域(フォルダ)です。
- 端末の内部ストレージやSDカードが該当します。
- (Android4.4以外は)SDカードを使える分、専用フォルダより容量が大きいです。
- アプリ外から見られて困るデータは置かないようにしましょう。

実際の利用例

- 端末のギャラリーに登録する画像、写真置き場
- エラーが出てアプリが落ちた時に送ってもらうエラーログファイル置き場(機密情報を含まないもの)

拡張ストレージのファイルアクセス

- Context#getExternalFilesDir()
  - 拡張ストレージ上の、アプリ用ディレクトリパスを取得する
- Context#getExternalCacheDir()
  - 拡張ストレージ上の、アプリ用キャッシュファイルを置くディレクトリパスを取得する

※Android 4.3までは、拡張ストレージへのファイルアクセスには「WRITE_EXTERNAL_STORAGE」「READ_EXTERNAL_STORAGE」のパーミッションが必要でしたが、Android 4.4以降はパーミッションなしでアクセスできるようになりました。

ファイルの書き込み例

```
File file = new File(getExternalFilesDir(null), "DemoFile.jpg");
InputStream is;
OutputStream os;
try {
   // 書き込むデータを用意する(ここでは画像リソース)
   is = getResources().openRawResource(R.drawable.balloons);
   // 書き込み用にファイルを開く
    os = new FileOutputStream(file);
    // 書き込みたいデータを全部読み込む
    byte[] data = new byte[is.available()];
    is.read(data);
    // データをファイルに書き込む
    os.write(data);
} catch (IOException e) {
    // 読み込み or 書き込みに失敗
    Log.w("ExternalStorage", "Error writing " + file, e);
}finally{
    // 終了処理
    if(is!=null){ is.close(); } // 入力ファイルを閉じる
    if(os!=null){ os.close(); } // 出力ファイルを閉じる
}
```

参考ソース: http://developer.android.com/reference/android/content/Context.html#getExternalFilesDir(java.lang.String)

復習

入出力ストリーム

- **InputStream**(入力ストリーム)
  - Javaでデータを読みとる(Inputする)時に必ず使われるクラス。
  - 子クラス:FileInputStreamなど。XxxInputStreamというクラス名なら大体InputStreamを継承しています。
- **OutputStream**(出力ストリーム)
  - Javaでデータを書き込む(Outputする)時に必ず使われるクラス。
  - 子クラス:FileOutputStreamなど。XxxOutputStreamというクラス名なら大体OutputStreamを継承しています。

- ファイルを開いて読み書きするときも、ネットワークと通信するときも使うものなので、ぜひ覚えておきましょう。
- HttpUrlConnectionなどを使って通信するときも、実は内部ではInputStream/OutputStreamを使って通信しています。

演習

演習

- SharedPreferenceを使ってみよう。
- アクセス制御が正しくできていることを確認しよう。
- 教科書51～56ページ
- 【手順】

1. ExportAppを実装して、MODE_PRIVATEで
   SharedPreferenceにデータが保存されることを確認する。
2. UseApp を実装して、ExportAppのSharedPreferencesに
   アクセスできないことを確認する。
3. ExportApp/UseAppを改変して、
   MODE_WORLD_READABLEだと外部からアクセスできること
   を確認する。

注意点

- (2)の注意
  - UseAppの「createPackageContext()」に指定する文字列はExportAppのパッケージ名です。
- (3)の注意
  - ExportApp/UseApp両方について、教科書通りgetSharedPreferences()に指定する文字列を「sample1」から「sample2」などに変更しましょう。
    ※SharedPreferencesは権限の上書きができず、同名のままだと初回保存時(=XMLファイル作成時)の権限(MODE_PRIVATE)のままになるため。
  - UseApp(読み取り側)はMODE_PRIVATEから変更しません。

チャレンジ演習

- 同じ方法で、端末に入っている他のアプリについても脆弱なSharedPreferencesがないか調査してみましょう。

【ヒント】

- SharedPreferencesは単純な名前が使われることが多いようです。
    - 例: pref, settings, settei, data, config などなど。
- 端末内に入っているアプリのパッケージ名はどうやって調べるか？
    - →教科書56ページ。または「パッケージ一覧 Android」などで検索。
- SharedPreferencesの中身を全部読み取るには、SharedPreferences#getAll() メソッドが利用できます。

確認

- ここまで実装できたら動きを確認してみよう。
- チャレンジ演習も余裕があったらやってみましょう。

ソースコードのポイント

- Context#getSharedPreferences(“プリファレンス名”, 読み書きモード)でインスタンスを取得する。
  - プリファレンス名＝保存するXMLファイル名になる。

- SharedPreferences#edit()で編集開始(Editorインスタンスを取得する)

- Editor#putXXX()でデータを編集する
  - putString(“key1”, “test”),putInt(“key2”,123), putBoolean(“key3”, true)など
  - 引数の１番目がキー(データの名前)、２番めが値(指定したキーに紐づくデータ)。

- Editor#apply()で編集された内容を書き込む(保存する)
  →apply()するまでファイルには保存されません。

- SharedPreferences#getXXX()で読み込む
  - getString(“key1”, null),getInt(“key2”,0), getBoolean(“key3”, false)など。
  - 引数の１番目がキー、２番めがデータが保存されてないときのデフォルト値。

SharedPreferences の使い方

```
public class MyActivity extends Activity {

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

        SharedPreferences pref = getSharedPreferences(“pref",
                MODE_PRIVATE);
        SharedPreferences.Editor editor = pref.edit();
        // プレイヤー名
        editor.putString("name", "山田太郎");
        // 年齢
        editor.putLong("age", 19);
        // ベストスコア
        editor.putInt("bestScore", 100);
        // 効果音再生: オン
        editor.putBoolean("sound", true);

        editor.apply(); // 保存
    }
}
```

MyActivity.java

書き込み

SharedPreferences#edit()

Editor#putXXX()

Editor#apply()

pref.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="name">山田太郎</string>
  <int name="age" value="19" />
  <long name=“bestScore" value="100" />
  <boolean name="sound" value="true" />
</map>
```

読み込み

SharedPreferences#getXXX()

ここまでのまとめ

- アプリ上のデータはどこかに保存しておかないとアプリ終了とともに消えてしまう。
- SharedPreferencesは簡単なデータの保存先。アプリの設定情報などを保存するのに便利。
- 大きなデータはデータベースに保存したり、アプリ専用フォルダに保存する。
- SharedPreferencesや、アプリ専用フォルダにファイルを作るときはMODE_PRIVATEとして、外部からアクセスを禁止しておく。
- アプリ外と共有するデータの置き場所には拡張ストレージが使える。

休憩(30分)

パーミッション

パーミッション

- Androidには、アプリがどんな権限(機能)を利用するかを宣言する仕組みがあります。これをパーミッションといいます。

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

- これまでの講義や演習の中でこんなおまじないを何度も書きましたね。
- これは「このアプリはインターネット通信を行う」というパーミッションです。
- 宣言しないで通信しようとするとSecurityExceptionという例外が発生して、アプリが落ちます。

パーミッションの例

permission名	意味
READ_CONTACTS	電話帳読み出し
WRITE_CONTACTS	電話帳書き込み
READ_CALENDAR	カレンダー読み出し
WRITE_CALENDAR	カレンダー書き込み
SET_ALARMS	アラーム設定
READ_SMS	SMS読み出し
WRITE_SMS	SMS書き込み
ACCESS_FINE_LOCATION	位置情報取得
WRITE_EXTERNAL_STORAGE	拡張ストレージへの書き込み
READ_PHONE_STATE	電話状態取得(端末ID取得)
BRICK	端末を文鎮にする

参考: http://developer.android.com/reference/android/Manifest.permission.html

パーミッションの作用

- Androidの様々な機能はパーミッション宣言をしたアプリにしか利用できなくなっています。
  → 一覧はAndroid SDKのreferenceを参照

- Google Playでアプリを公開すると、アプリのインストール時に、そのアプリがどういう権限を利用するかの一覧が表示されます。
  → ユーザは提示された権限に納得したら、アプリをインストールします。

パーミッションの確認

- パーミッションを設定していると
  Google Playからインストールしようと
  した時に、右のような画面がでます。

- ユーザはこの画面でアプリがどんなパー
  ミッションをAndroidManifest.xmlに
  設定しているか、確認することができま
  す。

- もし「緊急時の照明アプリ」として
  Google Playに出ているものが、電話帳
  機能へのアクセス機能を持っていたら信
  用してインストールしますか?

気をつけよう

- フラッシュライト(照明)アプリといいながら、情報を抜き出すというような不正事例は結構あります。

- 参考: http://appllio.com/20140623-5388-android-google-play-app-permission-deteriorate

- みなさんはそのような不誠実なアプリ開発をするのではなく、正面から良いアプリを作りましょう。

アプリ解析

アプリは簡単に抜き出せます

- 実はGoogle Playなどからインストールしたアプリの本体ファイルは、開発者機能で非常に簡単に抜き出すことができます。

- USBデバッグをONにした端末を繋ぎ、PCで以下のコマンドを使うだけです。

```
adb shell pm list packages -f
adb pull <apkファイルパス>
```

- アプリの本体ファイルは.apkという拡張子なので、apkファイルと呼ばれます。

アプリのデータも簡単に抜き出せます

- apkファイルはzip形式なので、拡張子をzipに変更することで簡単に展開できます。

- 展開した中にresフォルダの中には、画像ファイルなどが格納されています。

- ということは、アプリ内の画像などのリソースは簡単に抜き出せてしまうということです。

アプリのJavaソースコードも抜き出せます

- apkファイルを展開した中にあるclasses.dexというファイルがJavaコードをコンパイルして作られたバイナリファイルです。

- classes.dex はDalvik VM(Androidのアプリ実行環境)向けのコードですが、Javaのjarファイルに変換できます。

- さらに、jarファイルはJavaソースコードに戻せます。
(ただし、コメントなど元に戻らないものもあります)

演習

演習課題

- 以下のURLからダウンロードしたapkファイルを展開し、その中のclasses.dexをJavaソースコードに戻し、アプリ内に埋め込まれたログインパスワードとURLを調べてください
- ここにURL

手順

1. ファイルのダウンロードと展開
2. DEX ファイルを JAR ファイルに変換
3. JAR ファイルを解析して、Java ソースコードを確認

手順 1. ファイルのダウンロードと展開

i. enshu.apk をダウンロード

ii. enshu.apk を enshu.zip にリネーム

iii. enshu.zip を開き、その中から classes.dex ファイルがあることを確認します

手順 2. DEX ファイルを JAR ファイルに変換

i. 以下のサイトから dex2jar-2.0.zip をダウンロードし、展開します
http://sourceforge.net/projects/dex2jar/files/

ii. 展開したフォルダに先ほど確認した classes.dex をコピーし、コマンドプロンプトから d2j-dex2jar.bat を使って、classes.dex を jar ファイルに変換します

```
d2j-dex2jar.bat classes.dex
```

iii. classes-dex2jar.jar ファイルができているか確認します

手順 3. JAR ファイルの解析

i. 以下のサイトから jd-gui-windows-1.1.0.zip をダウンロードし、展開します
http://jd.benow.ca/

ii. jd-gui.exe を起動します

iii. classes-dex2jar.jar ファイルを、jd-gui.exe のウィンドウにドラッグ＆ドロップします

iv. jar ファイル内に含まれたクラスの Java ソースコードを確認してください

チャレンジ課題

- 端末にインストールされているGoogle製アプリを抜き出してください(例: Google Maps)
  ※アプリの抜き出し手順は、教科書を参考にしましょう

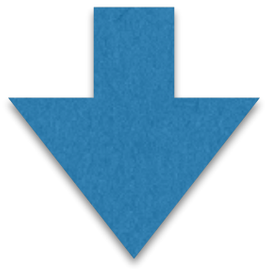
- アプリ内の画像ファイルを開いてみてください
- classes.dexをJavaソースコードに戻してください

確認

- URL、パスワードは分かりましたか？
- わかったら実際にapkをインストールして、ログインできるか試してみてください。
- また、余裕があればチャレンジ課題も挑戦してみましょう。
(チャレンジ課題はadbコマンドにパスを通す必要があります)

解析結果からわかること

- enshu.apkは難読化処理を何も行っていません。
- Google製のアプリでは、難読化処理が行われており、解析のヒントが減っています。

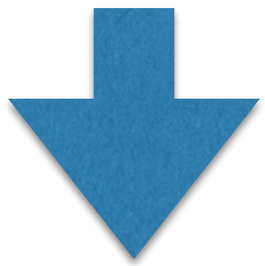


- この難読化処理は、開発環境に付属する ProGuardというツールによって行います。
- リリースするアプリでは、最低限 ProGuard を通しましょう。

参考: Proguardの有効化方法

- Android StudioとAndroid SDKは標準でProguardに対応しています。
- リリース時にProguardを有効化するには、AndroidManifest.xml を下記のように変更してください。

```
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}
```



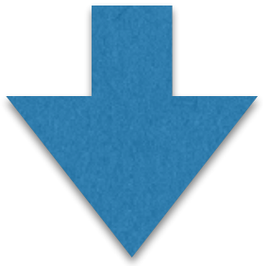
```
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}
```

Proguardの効能

- 不要な処理の除去、コードの最適化を行うことで、アプリのパフォーマンスがよくなります。
  (処理速度が早くなる、軽くなる)
- クラスやメソッドの名前、処理を短縮化して人には読みづらくします。
- ログ出力の削除を行い、解析のヒントを減らします。

元Javaソース

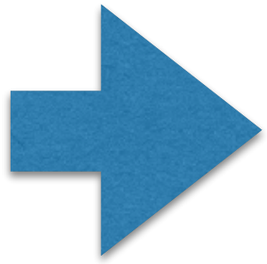
```
public class MySettings {
    public static final int getValue(){
        return 123 * 4467;
    }
}
```



コンパイルしただけ

```
public class MySettings
{
  public static final int getValue()
  {
    return 549441;
  }
}
```

Proguardで難読化



```
public class b
{
  public static final int a()
  {
    return 549441;
  }
}
```

Proguardの副作用

- Android SDKでは、よく使われる難読化条件を最初から設定してあるので、名前を変えてはいけないクラス、メソッド(Activtyクラス、onCreate()メソッドなど)は変換しないようになっています。

- ただし、全てをカバーしているわけではないので、まれに実装内容や利用しているライブラリによってはProguardを通すとアプリが動かなくなる場合もあります。

.

その場合は、Proguardの設定ファイルで、難読化してはいけないクラス等を設定する必要があります。

※今はProguardを有効にすると動かなくなる事があるという点だけ覚えておけば大丈夫です。
リリース時だけ動かない現象にぶつかったら、Proguardをオフにしてみたり、Proguardの設定方法で調べてみてください。

Proguardが守ってくれないもの

- Proguardは最低限の難読化を行ってくれますが、アプリ内の文字列、データ(画像などのリソース)などの暗号化はProguardでは行ってくれません。

- Proguardを実行すればソースは読みづらくはなりますが、enshu.apkにかかれていたような、アプリ内のパスワードやURLなどは暗号化されないので依然として比較的簡単に解析できてしまいます。

※どういう対策があるかは教科書72ページを
軽く読んでみてください。
(検索キーワード:DexGuard, Javaの文字列暗号化)

難読化・暗号化の意義

- ソフトウェアで難読化、暗号化しても地道にコードを解析していけば最終的には元データを取り出せます。
- しかし、難読化や暗号化をしておくことで、気軽な解析からは守ることができます。
- 暗号化を何重にも行うことによって、解析の手間がかかるようになるので、さらに解析しようとする人を減らせるかもしれません。

- 暗号技術の参考書籍
- 『暗号技術入門　秘密の国のアリス』結城浩 著
  - http://www.amazon.co.jp/dp/4797350997

参考

- 元のソースは
- https://raw.githubusercontent.com/ecoopnet/ti21015security/master/sources.zip
- においてあります。
- apkから取り出したソースと比較してみましょう。

Android アプリの
コンポーネントの
アクセス制限

Androidのコンポーネント

- Androidアプリケーションを構築するための４つの必要不可欠な構成要素のこと
- アクティビティ(Activity) … 画面
- サービス(Service)　… バックグラウンド処理
- コンテントプロバイダ(ContentProvider) … データ共有
- ブロードキャストレシーバ(BroadCastReceiver)
  … システム通知などの受信

参考: http://www.techdoctranslator.com/android/basics/fundamentals

コンポーネントは
外部アプリからもアクセスできる

- Activity
  - 自分のアプリAから他のアプリBの画面を開く
  - (メール送信画面、画像選択画面、etc.)
- Service
  - サービスに仕事を依頼する。
- ContentProvider
  - データの一覧を受取る、データを更新する。
- BroadcastReceiver
  - ブロードキャストインテントでBroadcastReceiverの機能を呼び出す。

Activityを呼び出す

- アプリ内のActivityを呼び出す。

```
Intent intent = new Intent(this, Hoge.class);
startActivity(intent);
```

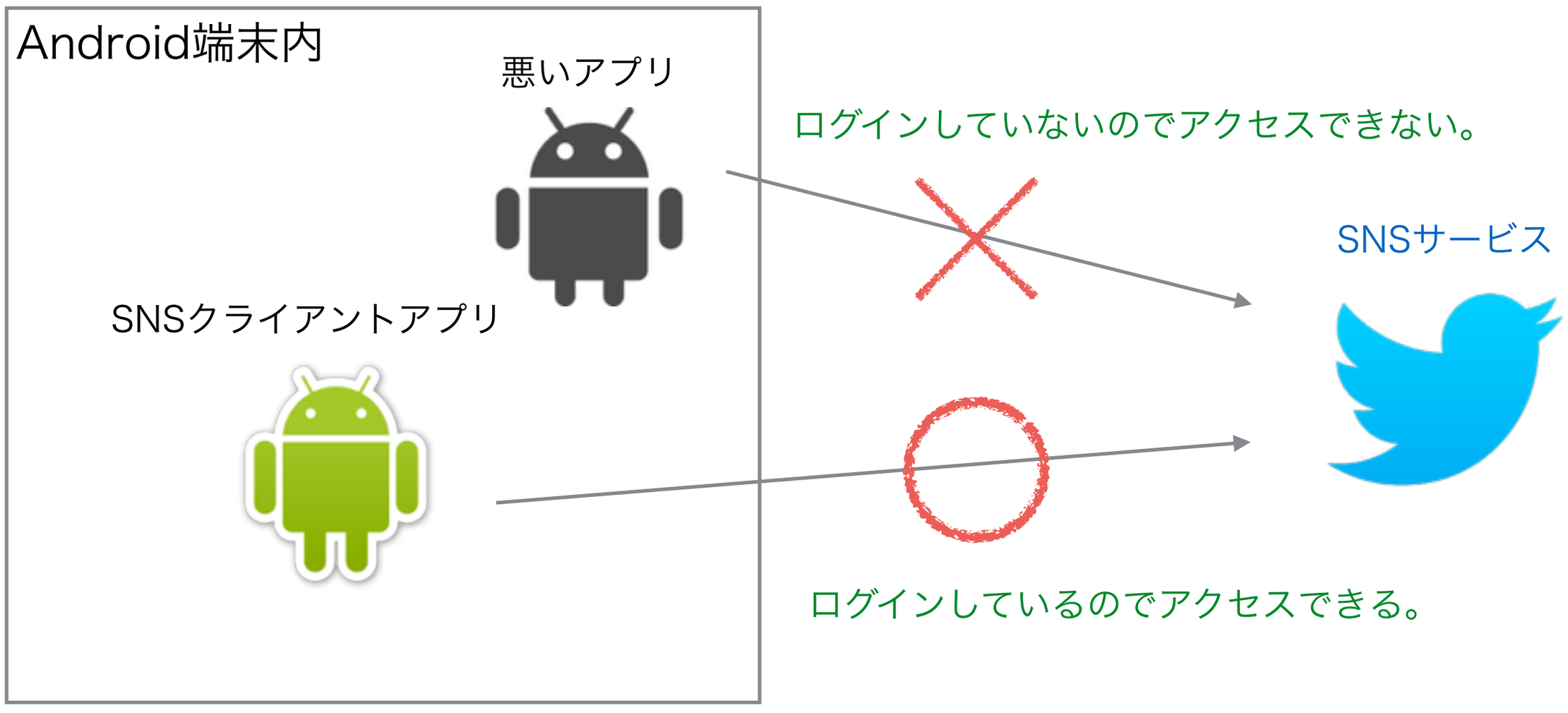
- 他のアプリのActivityを呼び出す一例。

```
Intent intent = new Intent();
intent.setClassName(“com.example.sample“, // 対象アプリのパッケージ名
        “com.example.sample.Hoge”); // クラス名
startActivity(intent);
```

- 制限をかけていないと、外部アプリからActivityは簡単に呼び出されてしまいます。

コンポーネントにセキュリティ不備があると…

- 例えばSNSクライアントアプリに問題があった場合、悪意あるアプリから無断でSNSクライアントアプリ経由で投稿されてしまうおそれがあります。



コンポーネントにセキュリティ不備があると…

Android端末内

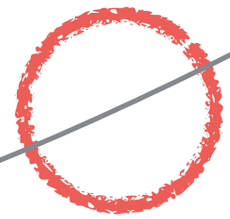
悪いアプリ

外部から認証なしで
投稿できる脆弱性を、
悪いアプリが悪用した！

セキュリティ不備のある
SNSクライアントアプリ

ログインしていないので
直接はアクセスできない。

SNSサービス



SNSクライアントアプリでは
ログインしているので
外部から不正にアクセスできてしまった。

コンポーネントの
簡単なアクセス制御方法

- AndroidManifest.xmlでコンポーネントのタグ(<activity>,<service>, <receiver>,<provider>)に、 android:exported=“false” という属性を設定すると非公開になり、アプリ外部からそのコンポーネントにアクセスできなくなります。
  安全のため、外部から使う必要のないコンポーネントは非公開にする習慣をつけましょう。
- 注意点として、Androidの仕様により、intent-filter があると、android:exported=“false"を指定していても外部に公開されてしまう場合があります。公開する必要のないActivityではintent-filterを指定しないようにしましょう。

参考: https://www.jssec.org/report/securecoding.html

コンポーネントの
簡単なアクセス制御方法

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>
```

コンポーネントの
高度なアクセス制御

- コンポーネントのうち、ContentProviderは外部アプリとの共有時のアクセス制限をさらに細かく制御できる仕組みがあります。

- 特定のアプリだけに公開する。
  - 自分が作った(署名した)すべてのアプリ間でアクセスを許可する。
  - 特定の１アプリからだけ、アクセスを許可する。

- この辺りはアプリが１つの間は使わない機能ですので、
  外部アプリからのアクセスが必要になったら調べてみましょう。
  (教科書60-64ページに詳しく書かれています)

ネットワーク通信のセキュリティ

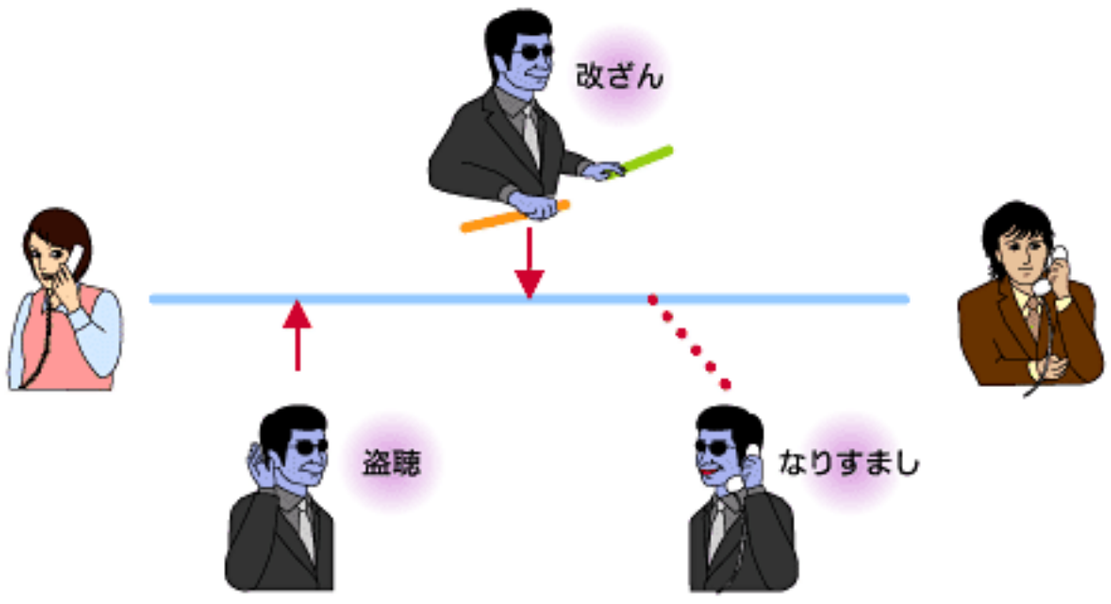
サーバとの通信

- スマートフォンアプリはサーバとの通信を行うことが多くなっています。
  例:
  - **ソーシャルゲーム**の**対戦機能、フレンド協力**
  - **SNSアプリ、チャットアプリ**の**メッセージやりとり、タイムライン表示や投稿**
  - **ニュースアプリ**の**記事の表示**

- このような、インターネット経由で誰かと繋がる機能がある場合はどこかのサーバとの通信が必要になります。

通信のセキュリティ

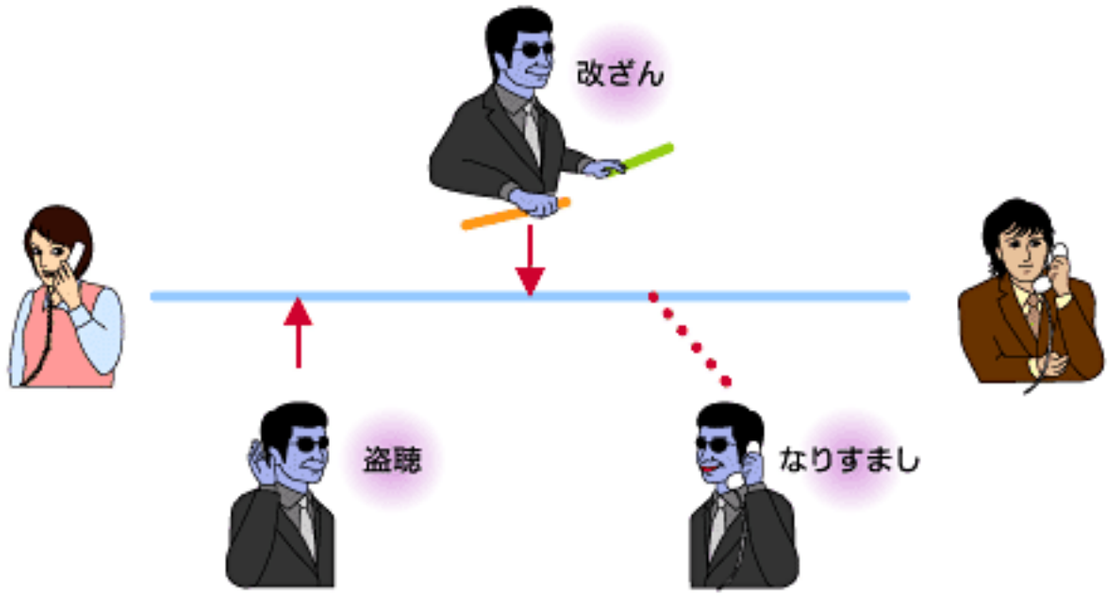
- セキュリティ上の問題として、インターネットの通信では常に悪意ある第三者に盗聴、改ざん、なりすましなどの不正を行われる危険があります。



参考: http://www.atmarkit.co.jp/ait/articles/0511/01/news132.html

盗聴

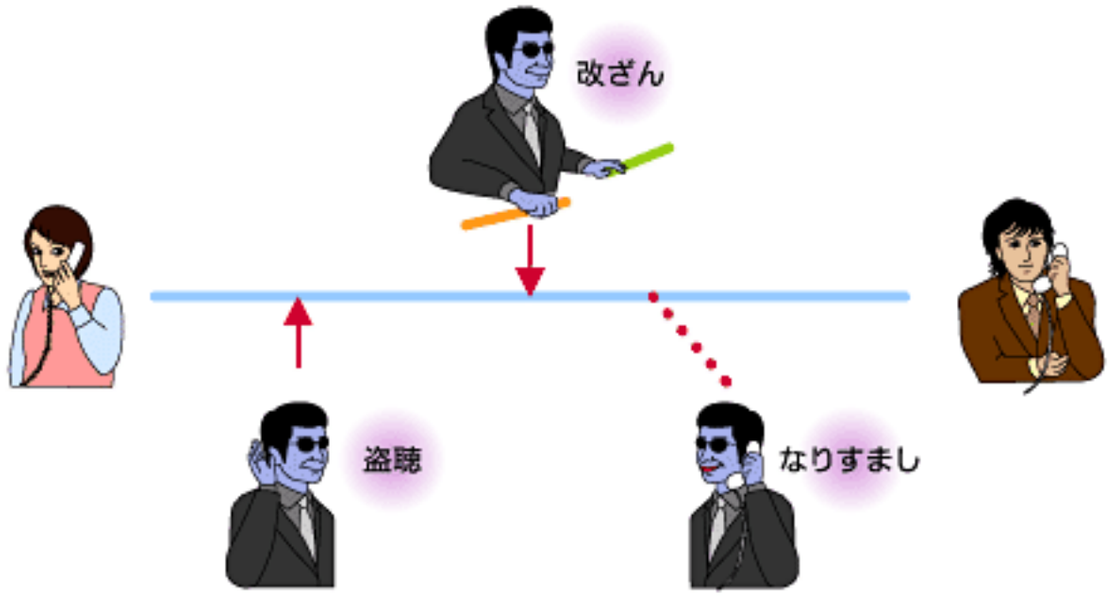
- 通信の盗聴をされると、たとえばクレジットカード番号が外部に漏れて勝手に買い物をされてしまったりします。
- アカウントのログインパスワードが盗聴されれば、アカウントハックにつながります。



参考: http://www.atmarkit.co.jp/ait/articles/0511/01/news132.html

改ざん

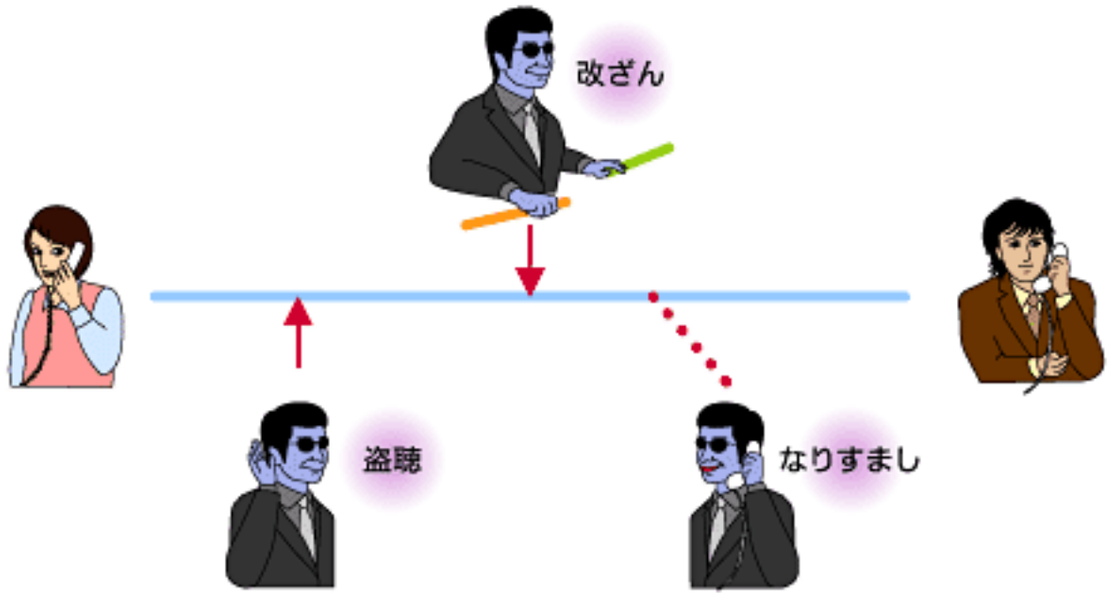
- 改ざんは、内容のすり替えです。
- たとえばネットショップのアプリの場合、１個の商品を買ったつもりが、第三者のいたずらで１００個の注文に書き換えられたりする可能性があります。



参考: http://www.atmarkit.co.jp/ait/articles/0511/01/news132.html

なりすまし

- なりすましは文字通り相手(サーバ)に「なりすます」事です。
- 例えば銀行のWebサイトになりすまされた場合、
  銀行のWebサイトBにつないでいるつもりが実はニセのWebサイトCに繋がっていた、そのままログインしようとしたら口座番号と暗証番号を盗まれてしまう、という問題があります。



参考: http://www.atmarkit.co.jp/ait/articles/0511/01/news132.html

安全な通信方法

- このように、ネットワークとの通信は悪意ある第三者から介入出来てしまうため、大事な通信は暗号化して、安全な通信にすることが必要になります。
- そのためにインターネットで広く使われているのが、SSL/TLSという仕組みです。

SSLとTLS

- SSL(Secure Socket Layer)
- TLS(Transport Layer Security)

- この２つはアプリとサーバの間を暗号化し、安全な通信経路にして盗聴・改ざん・なりすましを防ぐ通信経路を提供するためのプロトコルです。

- TLSはSSLを標準化した後継の通信仕様で、同じ機能なので、単にSSLとだけ呼ばれたり、SSL/TLSのようにセットで呼ばれる事が多いです。

SSL/TLSとHTTP

- SSL/TLSは暗号化・改ざん・なりすまし検出などセキュリティ上の機能だけを提供していて、実際のデータ通信には任意のTCPプロトコル(HTTPやFTPなど)が使用できます。

- 特にHTTPのSSL/TLS通信版はHTTPSプロトコルと呼ばれています。

https://www.google.co.jp/
のように、https:// から始まるURLではHTTPSプロトコル(TCP 443番ポート)を使用します。

アプリからのHTTPS通信

- AndroidではHTTPSプロトコルの通信を簡単に行うことが出来ます。

```
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
}
```

アプリからのHTTPS通信

- 前のスライドのソースコード通り、HTTPSの通信には**Https**UrlConnectionクラスを使用します。

- **Https**UrlConnectionクラスはHTTP用の**Http**UrlConnectionクラス(ネットワークの授業で実習した時に使ったものです)を継承しているので、
  実際にはHTTP通信と全く同じ方法(HttpUrlConnection)で通信できます。

- 同様に、ネットワークの授業で実習した「OkHttp」でも、HTTP/HTTPSをあまり気にせず通信することができます。

SSL/TLSの仕組み

1. クライアントがサーバに接続要求
2. サーバがクライアントに証明書を送る
3. クライアントがサーバ証明書を検証。正しければ乱数を、サーバ証明書に含まれる**公開鍵**で暗号化して送る
4. サーバは暗号化された乱数を秘密鍵で複合する
5. 乱数を共有できたので、その乱数を元に通信用の**共通鍵**を精製する

※概念説明のため簡略化しています

公開鍵暗号

- 暗号を受け取る相手の公開鍵で送信者が暗号化し、受信者は公開鍵に対応した秘密鍵で復号する
- 有名な暗号アルゴリズム： RSA、楕円曲線暗号

共通鍵暗号

- 暗号をやりとりするお互いが、同一の鍵を持ち、その鍵を使って暗号化と復号を行う
- 有名な暗号アルゴリズム： DES, 3DES, AES

共通鍵・公開鍵の
メリット・デメリット

- 公開鍵暗号
  - メリット：公開鍵では暗号化しかできないので、誰にでも渡していい(ネットで公開してもいい)
  - デメリット：暗号化・復号化が遅い
- 共通鍵暗号
  - メリット：暗号化・複合化が高速
  - デメリット：他人に鍵を知られないように、安全に交換しなければならない(鍵交換時に盗聴されてはいけない)

ハイブリッド暗号

- 共通鍵暗号と公開鍵暗号を組み合わせることで、お互いの弱点を補う
- 公開鍵ではこれからの通信で使用する共通鍵だけを送る(鍵交換する)
- 以降の通信は交換した共通鍵で行う
- SSL/TLSなどはこうのハイブリッド暗号を使用しています

※概念説明のため簡略化しています

SSLの
なりすましを防ぐ技術

- SSLでは証明書を用いて、正しい相手かどうかの身元を確認を行います。
- 悪い誰かがなりすまそうとしている場合でも、相手の証明書が正しくないことを検出できるので、安全に目的の相手に繋ぐことができます。
- ※Javaから実行した場合、証明書が異なるなどセキュリティ上の問題が発生した時はIOExceptionのサブクラスのSSLExceptionが発生します。
  通常の通信と同じくIOExceptionをキャッチしてエラー処理しておけば対応可能ですが、セキュリティを考慮するとSSLExceptionは専用の例外処理を行うことが望ましいです。

セキュリティのまとめ

- セキュリティ不備(脆弱性)のあるシステムは利用するユーザ側にとっても、運営側にとっても危険なものです。
- 作りたいアプリの仕様にセキュリティ上の問題がないか考えてみましょう。
- アプリ実装中の脆弱性のほとんどは、アクセス制限の不備が原因。正しいアクセス制限方法を覚えよう。
- 対策していないアプリは簡単に解析できる。中身を覗かれて当たり前と思い、難読化を忘れないようにしよう。また、見られて困るデータは暗号化するなどの対策しよう。
- 盗聴・改ざんをされて困る通信はSSLで通信経路を暗号化しよう。

まとめ(続き)

- 授業で紹介したものはあくまで一部です。
- 技術が進歩したり、新しいシステム出てくるたび、
  新しい攻撃方法や新しい脆弱性が生まれる可能性があります。

- セキュリティに限った話ではありませんが、開発者は常に学び続けていくことが大事です。

休憩(10分)