Google I/O 2012
Security and Privacy Android Apps

- 2012/6/29 (Day 3) 2:30PM - 3:30PM (1時間)
- http://youtu.be/RPJENzweI-A
Another privacy breach in the news...
At least it wasn’t your app this time!

Another Company Does Something Silly
posted 30 mins ago

In a surprising development, a company has accidentally committed a programming error which has compromised security. This comes after a series of other companies have done similar things. Users were quoted as saying, “Oh no, I’m going to uninstall this application immediately.”
Mobile devices are full of data...
Android protects access to sensitive data and device capabilities
Apps need to respect the data on Android devices

- People generally don’t like giving out their personal details to strangers
- Unscrupulous marketers want to mine mobile devices for data
  - User’s phone number and email address could be harvested for SPAM
  - Same with the people on their contact lists
- Criminals want to steal your money
  - Sending premium-rate SMS messages from your phone
  - Intercept two-factor authentication messages

$$$$
Insecure apps can grant unwanted access to data!

- When a user allows your app to access some aspect of their phone, they're trusting you with it
  - Please don’t let them down!
- If your app requests permissions, a security vulnerability in your app can grant other apps access to the protected data or component without permission
  - Storing personal data in a world-readable file
  - Exporting an unprotected content provider
  - Logging personal data in logcat logs
- It's not just other apps that you need to think about
  - Insecure wireless networks
  - Lost and stolen devices
Upload a privacy policy for your app
Let users know what you’re going to do with their data

Google Goggles
GOOGLE INC.

Open
Uninstall

DEVELOPER

Visit webpage
http://www.google.com/mobile/goggles

Privacy policy
http://www.google.com/policies/privacy
Developer account security
You don’t want other people to publish apps as you

Google Authenticator

Enter this verification code if prompted during account sign-in:

jlarimer@gmail.com

529643

Google play | ANDROID DEVELOPER CONSOLE

Manage user accounts

Pending Invitations

EMAIL | SHOW FINANCIAL DATA | INVITE SENT
kroot@gmail.com | Yes | 5/30/12

Users with access to this developer console

EMAIL | SHOW FINANCIAL DATA
jlarimer@gmail.com | Account owner
Application signing key

Your signing key is part of the identity of your app

Debug Key

App #1
(Debug)

Release Key

App #1
(Release)

App #2
(Release)

Same signing key means permissionLevel="signature" works!
Signing key security
Don't accidentally give out your key!
Signing key security
Don’t lose your key!

Google search results for "lost android release key".
Security architecture of Android

- **Browser Process**
  - Dalvik VM
  - Native Code
  - UID: app_0
  - open() permissions checked by kernel

- **CoolApp Process**
  - Dalvik VM
  - UID: app_12
  - Permission for Binder call to another app checked by system_server or app itself

- **CoolAddon Process**
  - Dalvik VM
  - Native Code
  - UID: app_19
  - WifiManager API call permissions checked by system_server

- **system server**
  - PackageManager
  - NetworkManager
  - ActivityManager
  - WifiManager
  - UID: system

**Linux kernel**
- Filesystem
- Wireless network driver
Security for your app

The application is in its own process sandbox.
- Dalvik gives you the freedom to add your own crypto implementations
- Reflection can be used to bypass scoping
  - `private` and `protected` may be ignored
- Native code can access and change data in the current process’s Dalvik VM - don't rely on the VM to provide security!

- For inter-process communication, there are protections:
  - Intent filters
  - Permissions
  - Signatures

CoolAddon Process

Dalvik VM
Native Code

UID: app_19
Typical application

Where’s the attack surface?
Typical application
Where’s the attack surface?
Protecting app components

App components and the AndroidManifest.xml file

- Accessible app components are declared in the **AndroidManifest.xml** file
  - Activities – `<activity>`
  - Services – `<service>`
  - Broadcast receivers – `<receiver>`
  - Content providers – `<provider>`
- Components specify what kind of **Intent** they accept with an `<intent-filter>` in the manifest
  - If a component has an `<intent-filter>` in the **AndroidManifest.xml** file, it’s exported by default
  - Content providers are the exception: they export data by default
- Don’t export app components unless you want other apps on the system to interact with your app
Limit access to components by external apps

This service has an intent filter so it must be explicitly marked as not exported

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.awesome">
  <application android:label="@string/app_name">
    ...
    <service android:name=".ServiceExample"
      android:exported="false">
      <intent-filter>
        ...
      </intent-filter>
    </service>
    ...
  </application>
</manifest>
```
Permissions for application components
Using permissions on exported components

- There are different permission protection levels available for apps:
  - **protectionLevel="normal"** – A lower-risk permission that gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user. This is the default protection level.
  - **protectionLevel="dangerous"** – A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user.
  - **protectionLevel="signature"** – Can be used to limit access to components to only apps signed with the same certificate.
Limit access to an exported component by permission

In this example an application signed with the same key can access the service.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.awesome">
    <permission android:name="com.example.awesome.EXAMPLE_PERM"
        android:label="@string/example_perm_desc"
        android:protectionLevel="signature" />

    <application android:label="@string/app_name">
        <service android:name=".ServiceExample"
            android:permission="com.example.awesome.EXAMPLE_PERM">
            <intent-filter>
                ...
            </intent-filter>
        ...
    </application>
```

Define a permission

Require the permission to access this service
Checking permissions in code
Sometimes you want finer-grained control over how permissions are enforced

- The **AndroidManifest.xml** should be used whenever possible to declare required permission.
- However, if it's not possible, there are other ways:
  - **Context.registerReceiver(...)** can be used to register a BroadcastReceiver dynamically:
    - There is a version of **registerReceiver(...)** which can be used to specify permission the broadcaster must hold for your dynamically-registered receiver to be invoked.
  - **Context.checkCallingPermission(...)** and **Context.enforceCallingPermission(...)** can be used in your source code to make sure the calling app holds the appropriate permission.
    - This can be used to implement fine-grained permissions if needed.

- Avoid the **confused deputy** problem:
  - If your app is using its granted permissions to respond to another app, check that the calling app has that permission as well.
Avoid being the confused deputy

WiFi Manager (Strict Sheriff)

Access Wifi

WifiControlApp

Confused Deputy

Access Wifi

WifiControlApp granted AttackerApp permission without checking

AttackerApp
Protecting Android apps from users
Don’t let users debug your apps

• **android:debuggable**
  - Disabled by default
  - Never leave this enabled in release code!
  - Allows a user to debug your app - even without source code
  - Users with physical access can run code as your app and access your app's data

```bash
jlarimer-macbookair:~ jlarimer$ adb shell
shell@android:/ $ run-as com.example.awesomeness sh
shell@android:/data/data/com.example.awesomeness $ id
uid=10060(app_60) gid=10060(app_60)
shell@android:/data/data/com.example.awesomeness $ ls files/
secret_data.txt
shell@android:/data/data/com.example.awesomeness $ cat files/secret_data.txt
SECRETS!
```
Storing data
Avoid exposing personal or protected data to other apps

• Protect personal data and data that requires a permission to access
  - Use MODE_PRIVATE for data files, shared preferences, and databases
    • openFileOutput(), openSharedPreferences(), and openOrCreateDatabase() create files in your app's private data directory
  - External storage (sdcard) is shared storage
    • Don't store personal or protected data on external storage without user consent

```
-rw-rw-rw-  app_53  app_53  8 2012-06-18 13:39 secret_data.txt
-rw-rw-rw-  app_53  app_53 81544 2012-06-18 21:43 private_info.txt
```

• You can't trust files that other apps can write to
  - Don't store code libraries that are world writable or on external storage
  - Don't store paths to code libraries in files that are world writable or on external storage
  - Don't process data from writable files in native code - memory corruption vulnerabilities could allow apps to run arbitrary code with your app's ID
Protecting data files
There are no good reasons to make your app’s private data files world readable

**Good:**

```java
FileOutputStream fos = openFileOutput("private_data.txt", Context.MODE_PRIVATE);
SharedPreferences prefs = getSharedPreferences("data", Context.MODE_PRIVATE);
```

**Bad:**

```java
FileOutputStream fos = openFileOutput("private_data.txt", Context.MODE_WORLD_WRITEABLE);
SharedPreferences prefs = getSharedPreferences("data", Context.MODE_WORLD_READABLE);
```
Data encryption doesn’t solve all problems
Encryption is not authentication!

\[
\text{EncryptedMessage} = \text{Encrypt}(K, "Login-OK=\emptyset")
\]

\[
\text{AlteredMessage} = \text{EncryptedMessage} \oplus \{\ldots, 0x31\}
\]

\[
\text{Plaintext} = \text{Decrypt}(K, \text{AlteredMessage}) = "Login-OK=1"
\]
Use a peer-reviewed library like **keyCzar**

Encryption is not authentication!

```java
java -jar KeyczarTool.jar create --location=/path/private.key \
    --purpose=crypt --name="My Server Key" --asymmetric=rsa

java -jar KeyczarTool.jar pubkey --location=/path/private.key \
    --destination=app/res/raw/server_pub.key
```

Crypter crypter = new Crypter(new AssetReader(R.raw.server_pub));
String ciphertext = crypter.encrypt("Secret message");
Rivest, Shamir, and Adleman took 42 tries to discover the RSA algorithm.

NIST received sixty-four entries by October 31, 2008.

NIST selected five SHA-3 finalists.
Protect network traffic
Attackers can eavesdrop on your app’s communications

• Assume that there's a bad guy reading all of your app's network traffic
  - Public WiFi networks can't be trusted
  - Rogue cellular base stations can intercept mobile network data traffic
• You can’t trust data coming from a server
  - Web servers can be compromised
  - Network traffic can be vulnerable to man-in-the-middle (MitM) attacks that insert malicious data into the network stream
Protecting network traffic
A man-in-the-middle attack can change your network traffic...
Protecting network traffic
A man-in-the-middle attack can change your network traffic...
Practice safe networking
Encrypt your network requests

- Best practice is to always encrypt network communications
  - HTTPS and SSL can protect against MitM attacks and prevent casual snooping
  - Server certificate validity is checked by default

```java
URL url = new URL("https://www.google.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
```

- Be very careful running code retrieved over the network
  - Use cryptographic signing for any DEX or native code libraries that you load dynamically
  - Better yet, don't run code from the network
Certificate pinning
If you don't completely trust the entire CA ecosystem...

Built-in CA List

SSLManager

TrustManager

HttpsURLConnection
Certificate pinning
If you don't completely trust the entire CA ecosystem...

Built-in CA List
CA #9
CA #85
CA #102
CA #73

My Certificate List
CA #85
My Cert

SSLManager
TrustManager
HttpsURLConnection

See the Android documentation on
HttpsURLConnection for example code
Using WebView

Don’t turn web problems into Android problems

- Watch out for cross-site scripting (XSS) and cross-site request forgery (CSRF) vulnerabilities if JavaScript is enabled on your WebView
  - JavaScript is disabled by default
  - If you run a web app in your Android app, you now have all of the security concerns of writing an Android app plus all of the security concerns with running a website

- `addJavascriptInterface()` is dangerous
  - Avoid exposing protected or personal data to a JavaScript interface
  - Server or network could be compromised, you can't trust the code
  - If you do use it, ensure that you're using HTTPS for the WebView
Minimize requested permissions

Users don't like when your app requests too many permissions...

Bad birdie

⭐⭐⭐⭐⭐ Ruben 6/14/12
Samsung Galaxy Nexus for an older version
Why are the birds stalking me? Now they want to know my location.

Permissions

⭐⭐⭐⭐⭐ Linford 6/14/12
HTC Sensation 4G for an older version
Not having location permissions
Minimize requested permissions

Users don’t like when your app requests too many permissions...

Insane permissions, poorly coded.

Mike 6/15/12
I love the concept but I really don’t know why this app needs SEND SMS MESSAGES, RECEIVE SMS, READ CONTACT DATA or WRITE CONTACT DATA. It doesn’t appear to send SMS or edit my address book or record videos, so there’s no need to ask for those permissions. The app looks like...

Uninstalled.

Zaib 5/2/12
Samsung Galaxy Note for an older version.
Do not understand why it needs GPS permission.? It’s suspicious.

From 5 to 1 star.

Byron 6/14/12
EeePad Transformer TF101 for an older version.
No way you’re tracking my location.

Bad birdie

Ruben 6/14/12
Samsung Galaxy Nexus for an older version.
Why are the birds stalking me? Now they want to know my location.

Permissions

Linford 6/14/12
HTC Sensation 4G for an older version.
Not having location permissions

Certainly does work...

Christine 5/15/12
Samsung Dart
However, the permissions are disturbing: “take pictures and videos: Allows the app to take pictures and videos with the camera. This allows the app at any time to collect images the camera is seeing.” And since the program also has network access, it could potentially take pictures of you or those around you and send them back to the programmer or who knows where else. “Your location coarse (network-based) location fine (GPS) location” For a flashlight? Why is this necessary? Last time I checked flashlights were not location-dependent. I’ll continue to carry my 4 sevens quark.:P “Read phone state and identity Allows the app to access...
Only request the permissions that your app requires
There are ways to access some Android capabilities without requesting permission

• Why minimize the amount of permissions your app requests?
  - One group of researchers found that 1/3 of apps request more permissions than they need
  - Security vulnerabilities can expose protected data
  - Users like apps that request few permissions

• Permissions aren't required if you launch an activity that has the permission
  - Getting a picture from the camera
  - Sending an SMS through the SMS app

• Permissions can be temporarily granted to apps by content providers
  - Letting the user pick a contact to share with your app
Get a camera pic without CAMERA permission
This prompts the user to take the picture, so they're in control of what your app gets

```java
// create Intent to take a picture and return control to the calling application
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

// create a file to save the image
fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE);
// set the image file name
intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);

// start the image capture Intent
startActivityForResult(intent, MY_REQUEST_CODE);
```
Start the SMS app with a filled-in destination and message

Doesn't require the SEND_SMS permission

Uri smsNumber = Uri.parse("sms:5551212");
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(smsNumber);
intent.putExtra(Intent.EXTRA_TEXT, "hey there!");
startActivity(intent);
Let the user choose a contact with ACTION_GET_CONTENT
Retrieve the selected contact data without requesting READ_CONTACTS

```java
Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
intent.setType(Phone.CONTENT_ITEM_TYPE);
startActivityForResult(intent, MY_REQUEST_CODE);

void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (data != null) {
        Uri uri = data.getData();
        if (uri != null) {
            try {
                Cursor c = getContentResolver().query(uri, new String[] {
                    Contacts.DISPLAY_NAME, Phone.NUMBER}, null, null, null);
            }
            ```
More minimizing requested permissions
More ways to reduce requested permissions

• Need a unique identifier?
  - TelephonyManager.getDeviceId() requires READ_PHONE_STATE permission.
  - Hardware IDs are a poor choice for identity anyway - see http://android-developers.blogspot.com/2011/03/identifying-app-installations.html
  - Settings.Secure.ANDROID_ID doesn’t require a permission, but still not perfect

• To identify an installation of your app
  - Generate a UUID when your app starts and store it in shared preferences:
    - String id = UUID.randomUUID().toString();
  - Use Android Backup Service to save the shared preferences to the cloud
  - See: https://developers.google.com/android/backup/
Device Administration access
Described for enterprise mobile device management (MDM) apps

- Device Administration API provides a lot of power, can be dangerous in the wrong hands
- Changing device security settings can have a serious impact on overall security
- Spend extra time auditing if your app can act as device administrator - you really don't want to leak these permissions!
Use Android Lint
Lint comes with the Android SDK and detects common programming errors

```
try {
    FileOutputStream fos = openFileOutput("secret_data.txt", Context.MODE_WORLD_WRITEABLE);
    fos.write("SECRETS!".getBytes());
    fos.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

There are cases where it is appropriate for an application to write world writeable files, but these should be reviewed carefully to ensure that they contain no private data, and that if the file is modified by a malicious application it does not trick or compromise your application.
Developer documentation on security
See these sites for more information on what we talked about today

  - Describes how various security features are implemented in Android
  - Teaches you how to write apps with security in mind
  - SDK documentation on the Android permission system

- **Application Security for the Android Platform**: *Processes, Permissions, and Other Safeguards*, Jeff Six, O'Reilly Media