Bits on Android security

Miroslav Svítok

About me

- I work as a penetration tester in Citadelo.
- I work (together with Dušan) at PhoneX - a mobile application for end to end encryption of calls, messages and files.

- Android security model
- Credentials storage
- Web view
- SSL/TLS and pinning

Android security model

Android security model

- Android = Linux-like OS
- Uses UID and GID in a different way than traditional Linux desktop or server
- Single physical user
- Sandboxing - **unique UID*** to each app, runs in a **separate process**
- App is given directory only it has permission to read/write

* exception - android:sharedUserId

Android security model - UIDs

- No /etc/passwd, system UIDs are statically defined.
- System daemons/apps UID 1000+ (e.g. SYSTEM user UID 1000, ROOT 0)
- Applications UID 10000+, automatically assigned

```
$ ps
 5786 u0_a71    289m S    {amsungapps.una2} com.sec.android.app.samsungapps.un
 5799 u0_a76    290m S    {.osp.app.signin} com.osp.app.signin
 5813 u0_a101   294m S    {nce.swype.input} com.nuance.swype.input
 5828 u0_a69    292m S    {moteNotiProcess} com.sec.spp.push:RemoteNotiProcess
 5869 u0_a105   366m S    {le.android.talk} com.google.android.talk
 5906 u0_a42    290m S    {id.partnersetup} com.google.android.partnersetup
```

`u0_a101 = UID 10101`

Android security model - application directory

- Directory is assigned by package name in /data/data
- E.g. /data/data/**com.google.android.youtube**

```
$ ls -la

drwxrwx--x    3 u0_a128  u0_a128       4096 Feb  9 16:05 cache
drwxrwx--x    2 u0_a128  u0_a128       4096 Mar  9 23:55 databases
drwxrwx--x    4 u0_a128  u0_a128       4096 Mar  9 23:56 files
drwxr-xr-x    2 system   system        4096 Feb 18 10:24 lib
drwx------    2 u0_a128  u0_a128       4096 Feb  9 16:05 no_backup
drwxrwx--x    2 u0_a128  u0_a128       4096 Mar 20 17:57 shared_prefs
```

Android security model

- Files in internal storage can be made readable/writeable for others
- MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE
- **@Deprecated (4.2+) and dangerous.**

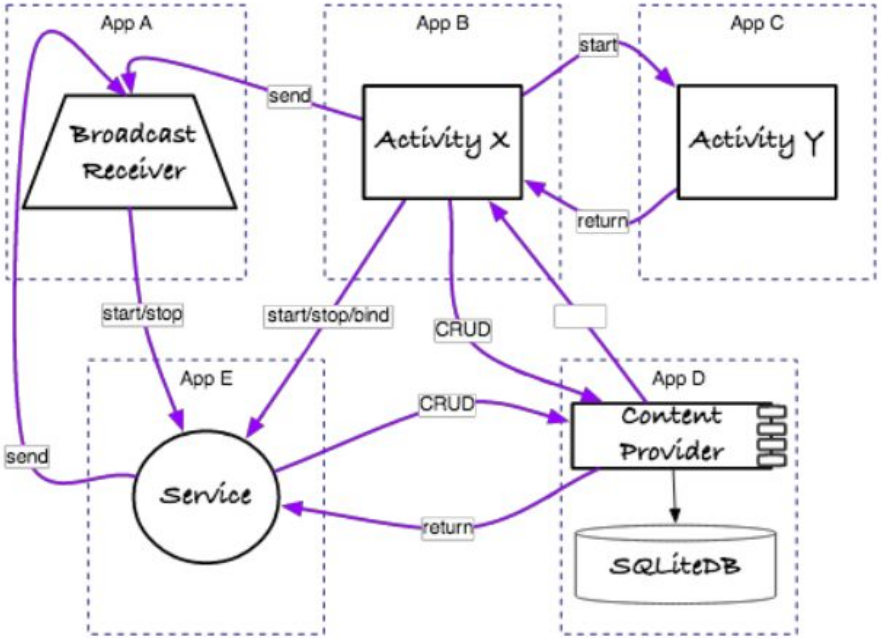
```
Context.getSharedPreferences(String name, int mode);
Context.openOrCreateDatabase(String name, int mode);
Context.openFileOutput (String name, int mode)
```

- Files in external storage are globally readable and writable!

- `android:installLocation: preferExternal` - apps on external storage

Communication with other applications??

- Interprocess communication! (IPC)
- Bypass processes isolation
- Implemented via Binder driver
- High-level abstraction:
  - *Intents*
  - *Messengers*
  - *ContentProviders*



Security of IPC - Permissions

- To control access, Android uses custom **Permissions** system**.**
- Requested in AndroidManifest.xml

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

- Assigned to app during installation (as supplementary GIDs).
- Permission protection levels:
  - Normal
  - Dangerous (=give access to user data or some form of control over the device, e.g. **SMSs**)
  - Signature
  - SignatureOrSystem

Security of IPC - Permissions

- Permissions handled by the PackageManager service
- Central database of installed packages
  - /data/system/packages.xml
- Cannot be changed or revoked by users without uninstalling app (<= Android 5.1)
- Android 6.0 + targetSdkVersion >= 23
  - Apps request permissions at runtime
  - Permission can be revoked

Security of IPC - Permissions

- Permissions can be enforced in a number of places
  - Start activity
  - Starting and binding a service
  - Sending and receiving broadcasts
  - Accessing a content provider
  - Making a system call
- High-level permission enforcement by `android:permission` attribute in AndroidManifest
- `Context.checkPermission (String permission, int pid, int uid)`

Security of IPC

- Minimize exposure of your ContentProviders, Services and Broadcast receivers.
- `android:exported="false"`
- Added `IntentFilter` often causes unintended exporting of features.
- You can require signature level permission - e.g custom one

```
android:permission="android.permission.ABC"

<permission
  android:name="android.permission.ABC"
  android:description="@string/permission_abc_desc"
  android:label="@string/permission_abc"
  android:permissionGroup="XYZ"
  android:protectionLevel="signature" />
```

Broadcast permissions

- Can be requested when sending (only app with perms)
- `Context.sendBroadcast(Intent intent, String receiverPermission)`
- `LocalBroadcastManager` - broadcast locally

Attacks

- **Broadcast theft** - data eavesdropped without knowing
- Ordered broadcasts (`Context.sendOrderedBroadcast`) can be stopped from propagating (depends on IntentFilter priority)
- **Activity hijacking** - malicious activity launched instead of intended (phishing)
- **Service hijacking -** app establishes connection with malicious service
- Stealing Special intents - granting permission
  - FLAG_GRANT_READ_URI_PERMISSION
  - FLAG_GRANT_WRITE_URI_PERMISSION
- **Intent spoofing**

Pending Intents

- Specifies an action that needs to be performed on behalf of your app by the other app in the future (NotificationManager, AlarmManager ...)
- Foreign app will execute Intent using **your app's** permission
- Can be stolen, be explicit:

```
//explicit (to MyService)
Intent intent = new Intent(context, MyService.class);
PendingIntent pi = PendingIntent.getService(getApplicationContext(), 0, intent, 0);
//implicit
Intent intent = new Intent("com.my.app.action")
PendingIntent pi = PendingIntent.getService(getApplicationContext(), 0, intent, 0);
```

- **CVE-2014-8609 Android Settings application privilege leakage vulnerability**

Credential storage

Credential storage options

- **KeyChain API** (Android 4.0+)
  - Import keys into system-wide credential storage
  - Can be used by different apps
- **KeyStore API** (Android 4.3+)
  - Support for app-private keys
  - Extraction prevention
    - Key material never enters application process
    - Key material may be bound to the secure hardware
- Backed by new Android JCA provider = AndroidKeyStore
- Entry types = CA certificate, user certificate, private key

Keystore implementation

- **keystore** service, runs as dedicated *keystore* user

```
# ls -la /data/misc/keystore/user_0
-rw------- keystore keystore        84    .masterkey
-rw------- keystore keystore       980    1000_CACERT_cacert
-rw------- keystore keystore       756    1000_USRCERT_test
-rw------- keystore keystore       884    1000_USRPKEY_test
-rw------- keystore keystore       724    10019_USRCERT_myKey
-rw------- keystore keystore       724    10019_USRCERT_myKey1
```

- .masterkey encrypted with AES-128, pass derived from screen unlock pass + random salt (PBKDF2, 8192 iterations)
- Other files - BLOB: **metadata || enc(MD5(data) || data)**

WebView

What is WebView?

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/webview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
```

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

WebView security

- Consumes web content.
- Limit its capability to minimum.
- Do not call setJavaScriptEnabled() if necessary.
- Security patches for WebView’s webkit not available on Android < 4.4

“Devices running platforms older than Android 4.4 (API level 19) use a version of webkit that has a number of security issues. As a workaround, if your app is running on these devices, it should confirm that WebView objects display only ***trusted*** *content.”*
- Android Developer Portal, Google

WebView - addJavascriptInterface()

- Enables to call Java functions via JS
- CVE-2012-6636 - execute attacker Java code via malicious JS
- **Vulnerable** code for Android < 4.2:

```
webView.getSettings().setJavaScriptEnabled(true);
webView.addJavascriptInterface(new JavaScriptInterface(), "jsinterface");

final class JavaScriptInterface {
  JavaScriptInterface () { }
   public void showToast() {                    // @JavascriptInterface
      Toast.makeText(mContext, “some toast”, Toast.LENGTH_SHORT).show();
   }
```

Good

```
<script>window.jsinterface.showToast();</script>
```

Bad

```
<script>
function execute(cmd){
  return window.jsinterface.getClass().forName('java.lang.Runtime')
    .getMethod('getRuntime',null).invoke(null,null).exec(cmd);
}
execute(['/system/bin/sh','-c','echo \"mwr\" > /mnt/sdcard/mwr.txt']);
</script>
```

SSL/TLS and Certificate pinning

Certificate pinning - Motivation

- Scenario: server side REST interface over SSL/TLS
- https://verysecure.com:4001
- Android by default trusts wide range of certificates for SSL/TLS
- **Settings -> Security -> Trusted credentials**
- **HttpsURLConnection** or **HttpClient** automatically validate against system's trusted certs.

Certificate pinning - Motivation 2

- Do we want to trust everybody?
- Sometimes CAs can be tricked or even **hacked** (Comodo - 03/2011, DigiNotar - 09/2011), or deliberately issue fraudulent cert to spy (French ANNSI - 12/2013).
- Single compromised CA can issue certificates for arbitrary domain (*.google.com?)
- Solution: **blacklist** or **whitelist**
- “Pinning is the process of associating a host with their expected X509 certificate or public key.“ - whitelisting approach

Certificate pinning - HowTo on Android

- Generate KeyStore (.bks) with your signing certificate
- KeyStore - load your custom keystore from the file system, initiate TrustManager with the KeyStore
- Create SSLContext
- or
- Pin against hash (SHA-256/512) of SubjectPublicKeyInfo (=public key + algorithm)
- **Keep it simple** = https://github.com/moxie0/AndroidPinning

Pinning Example

```
// Define an array of pins.  One of these must be present
// in the certificate chain you receive.  A pin is a hex-encoded
// hash of a X.509 certificate's SubjectPublicKeyInfo. A pin can
// be generated using the provided pin.py script:
// python ./tools/pin.py certificate_file.pem
String[] pins                 = new String[]
{"f30012bbc18c231ac1a44b788e410ce754182513"};
URL url                       = new URL("https://www.google.com");
HttpsURLConnection connection = PinningHelper.getPinnedHttpsURLConnection(context,
pins, url);

return connection.getInputStream();
```

The end

Thank you