Android Programming and Security

Dependable and Secure System

Andrea Saracino

andrea.saracino@iet.unipi.it

Part II

- Android System and Application Security.

Smartphone Security Issues

- Due to their large number of functionalities and connectivity interfaces, smartphones are liable to several security attacks.
- Possible attack vectors:
  - Bluetooth
  - Wi-Fi
  - Telephony (3G, SMS, phone calls…)

Smartphone Security Issues (2)

- Targets of the attack:
  - Private data
    - SMS messages
    - Contacts
    - IMEI code
    - Username and Passwords (social network, home banking).
  - User Money
    - Leaking credit from SIM card.
    - Hidden subscription to premius services.

Attack Methods

- Privilege Escalation: an application communicate with another one with higher privileges, to perform a malicious unauthorized operation.
- Trojan: hidden malicious code that sneakily performs malicious operations.
- Social Engineering: attempts to force the user to unveil private data.
- Eavesdropping.

Standard Security Mechanism

- Every mobile phone (non only smartphone) send and receive encripted traffic.
- The keys to encrypt/decript data are stored in the SIM card.
- Internet connection may use https and certificates.
- Basic integrity checks of the OS and application signature verification are provided by almost all mobile OSs.

Standard security mechanisms

- Generally they only manage to limit eavesdropping.
- Ineffective against the other attacks…
- Need of specific solutions!

Native Android Security Mechanisms

- Sandboxing (Isolation)
  - Dalvik Virtual Machine
  - Linux Kernel
- Access Control
  - Permission System

Sandboxing

- Dalvik Virtual Machine act as a sandbox for Android applications.
- Each application can perform all of its operations inside the virtual machine.
- Each application operates like it there are no other applications running on the device.
- Application cannot communicate directly.

Isolation

- Every Android application has a different Linux User ID.
- Different storage space: an application cannot modify files of other applications.
- Application execution is interference-free.
- This should avoid the privilege escalation attacks.
- Android applications are normal Linux user without root privilege: an application cannot modify system files.

Access Control

- Critical resources and access to dangerous operations, in Android smartphones are protected by access control mechanisms.
- Examples of Critical Resources:
  - Contact Lists, SD Card, Smartphone Data, Accounts, SMS inbox/outbox…
- Examples of Critical Actions:
  - Send SMS messages, start phone call, access the internet, turn on/switch off connectivity interfaces…

Permission System

- An Android application that will access a critical resource, or will perform a protected operation, have to ask the permission to do so.
- Permissions can be seen like a declaration of intent. The application developer declares that the application want to perform a critical operation.

Permissions in Manifest

- Permissions are declared by developer in the manifest file, using a specific XML tag:

<uses-permission android:name="*string*" />

- Android defines 120 permissions, identified by the name: *android.permission.Permission*

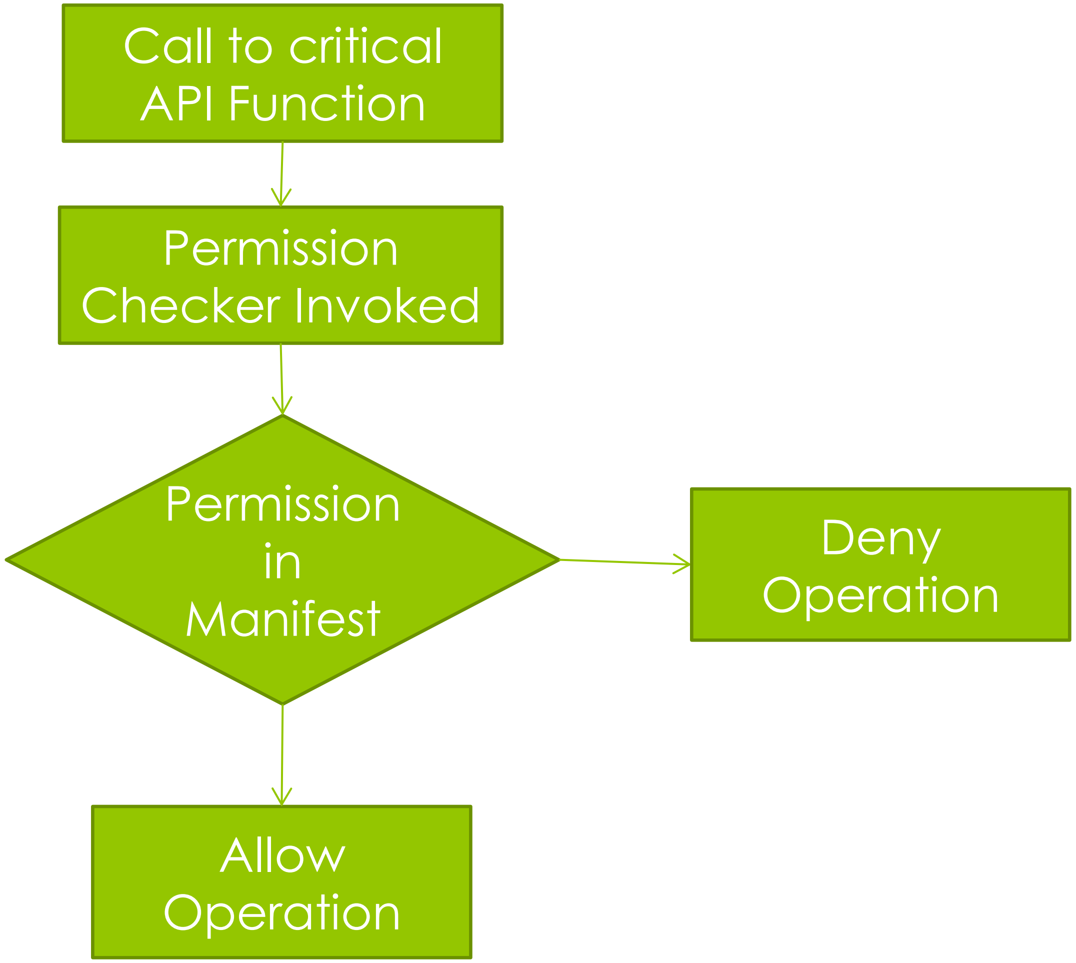
Permission Levels

- There are four levels of permissions:
  - Normal: Related to non-dangerous resources. Example: VIBRATION.
  - Dangerous: Related to critical resources and costly services. Example: SEND_SMS.
  - Signature: Permissions that can be required only by applications developed by the same developers.
  - Signature or System: Like signature, but can be required also by Android native applications.

Permission Checker

- The permission checker is the component that verifies at runtime, if an application that is going to perform a critical operation, has declared the related permissions.
- If the permission has been declared the operation is allowed, otherwise the operation is denied.

Permission Verification



Permission Verification (2)

- The API functions that perform critical operations call the permission checker before doing any operation.
- The permission checker verifies that the application has the permission related to critical operation that the function will perform.
- If the permission has not been declared the operation is denied. The function throws a **security exception** for missing permission and no return is given from the function.
- Generally the exception is not caught by the developer or the missing return is not handled, causing the application to crash.

Static Permission VS Dynamic Verification

- Permissions are declared statically in manifest files. Verification is performed dynamically.
- It is possible that a developer call in the Java code a critical function without asking for the permission in the manifest file.
  - Programming error. No warning are raised! When including a potentially critical function control the API documentation to see the required permissions.

Creating Permissions

- A developer can define permissions to protect an application or components of an application.
- If some operations performed by an application are critical on the side of security they should be protected.
- <permission android:description="string resource"
  android:icon="drawable resource"
  android:label="string resource"
  android:name="string"
  android:permissionGroup="string"
  android:protectionLevel=["normal" | "dangerous"
  |"signature" | "signatureOrSystem"] />

Creating Permissions (2)

- With a permission is possible to protect:
  - An activity
  - A service
  - An intent filter
  - The whole application.
- The protected resource has to be specified in the android:name field of the permission.

Malware

- Starting from 2009 mobile platforms, especially Android, have become target of specific attacks.
- The largest increase in malware number targeting Android devices has been observed in 2011.
- Two main types: Trojans and Rootkits.

Trojans

- Hidden malicious code in good looking applications. These malware comes as normal applications that works fine (videogames, contact magers…). However in background these applications stealthy perform malicious behaviors.
- Example of malicious behaviors:
  - Stealthy sent SMS messages.
  - Recording phone calls.
  - Send private data to external servers.
  - Subscription to premium services.

Simple but Effective

- These malware are hardly identified, they perform normal actions in a malicious way.
- Example: Sending an SMS message is not a malicious action by itself. It becomes malicious if it is done stealthily, with undesired costs, or if the messages are sent to premium rate numbers.

Rootkits

- Malicious applications that take control of the device, generally exploiting some system weaknesses.
- The application take root privileges and then perform strong attacks like:
  - Downloads and installs other malicious applications.
  - Delete user files.
  - Steal private data.

Root Access

- Shell and other applications are users without root privileges.
- The command **SU** (switch user) has been disabled in the modified kernel version used by Android.
- Rooting an Android device simply means to enable again the SU command.
- The rooting method is different for each device. Generally is performed through a buffer overflow, where the instructions to install the SU command are pushed in the execution stack, exploiting system (hardware or software) vulnerabilities.

Detect Malware With Permissions

- Trojans can be recognized, sometimes, because they ask for non-justified permissions:
  - Why a videogame should ask to send SMS messages or perform phone calls?
  - Once the permission has been granted, the malware will legally perform the malicious behavior of sending messages.

Weaknesses

- **Too coarse grained**. For example the SEND_SMS permission does not give any specification on the number of outgoing SMS messages, neither the destination number is specified.
- Accept all or deny. An application may ask for several permissions. If a single permission is not accepted, the application is not installed.
- Rootkits are able to take control without asking for permissions.

Weaknesses

- Who is really enforcing the security policy of permissions?
- **The user**!!
- When user downloads an application and wants to install it, a list of the required permission is shown. Then the user chooses to accept them all or stop the installation.
- According to some recent surveys more than 60% of the users do not even know what permissions are.

Permission Overdeclaration

- The problem of permission overdeclaration is dual to the error of forgetting to declare a needed permission.
- A developer declare in the manifest more permissions than those effectively required by the application.
- This error is extremely common for at least two reasons:
  - Enthusiast developers with limited experience.
  - Name and description of permissions confusing.
- The effect of permission overdeclaration is decreasing the attention that users put in reading permissions.

Some Research Solutions

- Allow the user to choose a subset of permissions for each application.
- Tracking outgoing information and avoiding requests and transmission of private data.
- TISSA: modification of the permission checker to avoid security exception and providing empty or bogus results.

Some Research Solutions (2)

- Definition of policies to avoid dangerous coupling of permissions.
  - An application that can access user private data, should not be allowed to connect to the network.
  - An application cannot send SMS when the user is not active.

Some Research Solutions (3)

- Profiling device, user and application behaviors.
- Use statistics (Chi-Squared), probability (Markov Chain) or computational intelligence (ANN, Classifiers) to detect anomalies in the execution.
- Extremely effective in detecting rootkits and trojans.

Some Research Solutions (4)

- Static classification of Android application analyzing declared permissions and other metadata.
- Assigning a threat value to each permission and thus to each application.
- Avoid permission overdeclaration, rewarding application that declare as less permissions as possible.