Android Security Mechanisms

Lecture 5

Security of Mobile Devices

2018

Android Security Mechanisms

Application Sandbox

Android Permissions

Signing Applications

Bibliography

Outline

Android Security Mechanisms

Application Sandbox

Android Permissions

Signing Applications

Bibliography

Security Goals

- Protect application and user data
- Protect system resources
- Isolate app from the system, other apps and the user

Key Security Features

- Linux kernel security
- Application sandbox
- Signed applications
- Permissions
- Secure IPC

Outline

Android Security Mechanisms

Application Sandbox

Android Permissions

Signing Applications

Bibliography

Sandboxing

- Mechanism based on UIDs
  - Isolate applications
  - Unique UID assigned to each application at installation time
  - Dedicated process running as that UID
  - Dedicated directory - only that UID has rwx permissions
- Process-level and file-level sandbox
- Enforced at kernel-level

Application Files Permissions

- Each app - dedicated data directory
  - rwx permissions only for that app UID/GID
  - Other apps cannot access those files
- MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE flags
  - Gives read or write access to files
  - Deprecated from Android 4.2

UIDs

- Well-defined UIDs for system services and daemons
- User `root` UID 0
  - Very few daemons under root UID 0
- User *system* UID 1000
  - Special priviledges
- UIDs for system services start at 1000
- App UIDs start at 10000

Shared UID

- Apps with the same UID
  - Share files
  - Run in the same process
- Frequently used by system apps
  - Not recommended for non-system apps
- Implementation:
  - Signed with the same code signing key
  - `sharedUserId` attribute of `<manifest>`

Outline

Android Security Mechanisms

Application Sandbox

Android Permissions

Signing Applications

Bibliography

Android Permissions

- By default, applications cannot perform operations to impact other apps, the OS or the user
- Permission - the ability to perform a particular operation
- Built-in permissions documented in the platform API reference
  - Defined in the `android` package
- Custom permissions - defined by system or user apps
- `pm list permissions`

Android Permissions

- Defining package + .permission + name
  - `android.permission.REBOOT`
  - `com.android.laucher3.permission.RECEIVE_LAUNCH_BROADCASTS`
- Apps request permissions in `AndroidManifest.xml`

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

- Until Android 6: granted at install time; cannot be changed or revoked
- From Android 6.0: apps request permissions at runtime; user can revoke permissions

- Permissions handled by the PackageManager service
- Central database of installed packages
  - `/data/system/packages.xml`
- Programatically access package information from `android.content.pm.PackageManager`
  - `getPackageInfo()` returns `PackageInfo` instance

- A permission can be enforced in a number of places
  - Making a call into the system
  - Starting an activity
  - Starting and binding a service
  - Sending and receiving broadcasts
  - Accessing a content provider

Permission Protection Levels

- Potential risk and procedure to grant permission
- Normal
  - Low risk
  - Automatically granted without user confirmation
  - `ACCESS_NETWORK_STATE`, `GET_ACCOUNTS`

Permission Protection Levels

- Dangerous
  - Access to user data or control over the device
  - Requires user confirmation
  - `CAMERA, READ_SMS`

- Signature
  - Highest level of protection
  - Apps signed with the same key as the app that declared the permission
  - Built-in signature permissions are used by system apps (signed with platform key)
  - `NET_ADMIN, ACCESS_ALL_EXTERNAL_STORAGE`

Permission Protection Levels

- SignatureOrSystem
  - Apps part of system image or signed with the same key as the app that declared the permission
  - Vendors may have preinstalled apps without using the platform key

- All permissions belong to permission groups
- Dangerous permission groups
- Examples of dangerous permission groups:
  - Calendar, Camera, Contacts, Location, Phone, SMS, Sensors, Storage, Microphone

Permission Groups

- Until Android 5.1:
  - Permission groups are requested at install time (not the individual permissions)
- On Android 6.0:
  - If there is no other permission in that group, it requests the user's confirmation for that permission group
  - If there is another permission in that group already granted, it does not request any confirmation

Kernel-Level Enforcement

- Access to regular files, device nodes and local sockets managed by the Linux kernel, based on UID, GID
- Permissions are mapped to supplementary GIDs
- Built-in permission mapping in `/etc/permission/platform.xml`

Kernel-Level Enforcement

- Example:
  - INTERNET permission associated with GID inet
  - Only apps with INTERNET permission can create network sockets
  - The kernel verifies if the app belongs to GID inet

Framework-Level Enforcement

- Static permission enforcement
  - System keeps track of permissions associated to each app component
  - Checks whether callers have the required permission before allowing access
  - Enforcement by runtime environment
  - Isolating security decisions from business logic
  - Less flexible

Framework-Level Enforcement

- Dynamic permission enforcement
  - Components check to see if the caller has the necessary permissions
  - Decisions made by each component, not by runtime environment
  - More fine-grained access control
  - More operations in components

Static Enforcement

- An app tries to call a component of another app - intent
- Target component - `android:permission` attribute
- Caller - `<uses-permission>`
- Activity Manager
  - Resolves intent
  - Checks if target component has an associated permission
  - Delegates permission check to Package Manager
- If caller has necessary permission, the target component is started
- Otherwise, a SecurityException is generated

Dynamic Enforcement

- Helper methods in `android.content.Context` class to perform permission check
- `checkPermission(String permission, int pid, int uid)`
  - Returns `PERMISSION_GRANTED` or `PERMISSION_DENIED`
  - For root and system, permission is automatically granted
  - If permission is declared by calling app, it is granted
  - Deny for private components
  - Queries the Package Manager
- `enforcePermission(String permission, int pid, int uid, String message)`
  - Throws SecurityException with message if permission is not granted

Activity and Service Permission Enforcement

- Permission checks for activities
  - Intent is passed to `Context.startActivity()` or `startActivityForResult()`
  - Resolves to an activity that declares a permission

Activity and Service Permission Enforcement

- Permission checks for services
  - Intent passed to `Context.startService()` or `stopService()` or `bindService()`
  - Resolves to a service that declares a permission
- If caller does not have the necessary permission, generates SecurityExceptions

Content Provider Permission Enforcement

- Protect the whole component or a particular exported URI
- Different permissions for reading and writing
- Read permission - `ContentResolver.query()` on provider or URI
- Write permission - `ContentResolver.insert()`, `update()`, `delete()` on provider or URI
- Synchronous checks

Broadcast Permission Enforcement

- Receivers may be required to have a permission
  - `Context.sendBroadcast(Intent intent, String receiverPermission)`
  - Check when delivering intent to receivers
  - No permission - broadcast not received, no exception

Broadcast Permission Enforcement

- Broadcasters may need to have a permission to send a broadcast
  - Specified in manifest or in `registerReceiver`
  - Checked when delivering broadcast
  - No permission - no delivery, no exception
- 2 checks for each delivery: for sender and receiver

Add Permission in Manifest

- On all Android versions

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.smd">

    <uses-permission android:name="android.permission.SEND_SMS"/>
    <!-- other permissions go here -->

    <application ...>
        ...
    </application>
</manifest>
```

Check for Permissions

- Dangerous permissions must be granted by the user
- Check if app has dangerous permission before performing operation
  - Permissions can be revoked from Android 6
- `ContextCompat.checkSelfPermission()`
  - Returns `PERMISSION_GRANTED` - operation can be performed
  - Returns `PERMISSION_DENIED` - permission must be requested

Request Permissions

- When `checkSelfPermission()` returns `PERMISSION_DENIED`
- Provide explanation for permission request
- Method `ActivityCompat.requestPermissions()`
  - Permission
  - Request code
- App needs to request every permission even if user grants whole group

Request Permissions

- Dialog box shown by the system
  - Requests permission group
  - Cannot be changed by the app
  - Explanation provided before requesting permissions
- Asynchronous
  - Response received in callback

Request Permissions - Example

```
if (ContextCompat.checkSelfPermission(thisActivity,
        Manifest.permission.READ_CONTACTS)
    != PackageManager.PERMISSION_GRANTED) {

    // Permission is not granted
    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
            Manifest.permission.READ_CONTACTS)) {

        // Show an explanation to the user *asynchronously* -- don't block
        // this thread waiting for the user's response! After the user
        // sees the explanation, try again to request the permission.

    } else {

        // No explanation needed; request the permission
        ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);

        // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
        // app-defined int constant. The callback method gets the
        // result of the request.
    }
} else {
    // Permission has already been granted
}
```

Handle Permissions Request Response

- User responds -> system calls `onRequestPermissionsResult()` callback
  - App must override this method to receive results
  - Request code, permissions and results received as parameters
  - Check request code
  - Check if permission is granted

Handle Permissions Request Response

- Permission granted
  - Do permission-related task
- Permission denied
  - Disable functionality
  - Announce user

Handle Permissions Request Response - Example

```
@Override
public void onRequestPermissionsResult(int requestCode,
        String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

                // permission was granted, yay! Do the
                // contacts-related task you need to do.

            } else {

                // permission denied, boo! Disable the
                // functionality that depends on this permission.
            }
            return;
        }

        // other 'case' lines to check for other
        // permissions this app might request.
    }
}
```

Custom Permissions

- Declared by apps
- Checked statically by the system or dynamically by the components
- Declared in `AndroidManifest.xml`

Custom Permissions - Example

```
<permission-tree
        android:name="com.example.app.permission"
        android:label="@string/example_permission_tree_label" />

<permission-group
        android:name="com.example.app.permission-group.TEST_GROUP"
        android:label="@string/test_permission_group_label"
        android:description="@string/test_permission_group_desc" />

<permission
        android:name="com.example.app.permission.PERMISSION1"
        android:label="@string/permission1_label"
        android:description="@string/permission1_desc"
        android:permissionGroup="com.example.app.permission-group.TEST_GROUP"
        android:protectionLevel="signature" />
```

Android Security Mechanisms

Application Sandbox

Android Permissions

Signing Applications

Bibliography

Signing Applications

- Identify the developer of the application
- All apps must be signed
- Unsigned apps rejected by Google Play and package installer
- Bridge between Google's trust in the developer and the developer's trust in the app
- Developer is accountable for the behavior of the app

Certificate

- Each apk signed with a certificate
- Generated using the developer's private key
- Identifies the developer of the application
- Can be self-signed
- Update allowed only if the certificate matches

- Package Manager verifies signature
- At installation time
- Verfication uses the public key in the certificate included in the apk
- Grants package integrity
- System applications signed with the platform key

Same key

- Possible shared UID
- Signature protection level permission
  - Granted to apps signed using the same key only
  - Different sandboxes and UIDs

Signing Schemes

- v1 scheme - based on JAR signing
- v2 scheme - APK SIgnature Scheme v2
  - From Android 7
- For compatibility - sign with both schemes
  - Android $>=$ 7 check v2 signature
  - Android $<$ 7 check v1 signature

Schema v1

- Does not protect parts of the apk
  - ZIP metadata
  - Verifier checks data structures
  - Discards data not covered by signature
  - Attack surface
- Verifier has to uncompress all compressed entries
  - Time and memory consuming

Schema v2

- APK is hashed and signed
  - => APK Signing Block
  - Inserted into APK
- Verification:
  - Treats APK as a blob
  - Checks signature across entire file
  - Any alteration of the APK invalidates signature
  - Faster
  - Detects more unauthorized alteration

Android Security Mechanisms

Application Sandbox

Android Permissions

Signing Applications

Bibliography

Bibliography

- Android Security Internals, Nikolay Elenkov
- https://source.android.com/security/
- https://source.android.com/security/apksigning/
- https://developer.android.com/guide/topics/permissions/overview.html

Keywords

- Signed applications
- Shared UID
- Permissions
- Protection levels
- Static enforcement
- Dynamic enforcement
- Custom permissions