Android Permissions

Cryptographic Providers

Network Security

Bibliography

Keywords
Android Permissions

Cryptographic Providers

Network Security

Bibliography

Keywords
Android Permissions

- A string
- 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`
- Defining package + .permission + name
  - `android.permission.REBOOT`
  - `com.android.launcher3.permission.RECEIVE_LAUNCH_BROADCASTS`
Apps request permissions in `AndroidManifest.xml`

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

- Assigned to apps at install time by the package manager service
- Central database of installed packages
  - `/data/system/packages.xml`
- Programatically access package information from `android.content.pm.PackageManager`
  - `getPackageInfo()` returns `PackageInfo` instance
- Cannot be changed or revoked without uninstalling app
Permission Protection Levels

- Potential risk and procedure to grant permission
- Normal
  - Low risk
  - Automatically granted without user confirmation
  - ACCESS_NETWORK_STATE, GET_ACCOUNTS
- Dangerous
  - Access to user data or control over the device
  - Requires user confirmation - accept or cancel installation
  - CAMERA, READ_SMS
Permission Protection Levels

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

- **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
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

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

- **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
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
Static Enforcement

- An app tries to call a component of another app - intent
- Target component - android:permission
- 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
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
- 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

- 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
Custom Permissions

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

```
<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" />
```
Outline

Android Permissions

Cryptographic Providers

Network Security

Bibliography

Keywords
Java Cryptography Architecture (JCA)
- Extensible cryptographic provider framework
- Set of APIs - major cryptographic primitives
- Applications specify an algorithm, do not depend on particular provider implementation

Cryptographic Service Provider (CSP)
- Package with implementation of cryptographic services
- Advertises the implemented services and algorithms
- JCA maintains a registry of providers and their algorithms
- Providers in a order of preference

Service Provider Interface (SPI)
- Common interface for implementations of a specific algorithm
- Abstract class implemented by provider
JCA engines provide:
  ▶ Cryptographic operations (encrypt/decrypt, sign/verify, hash)
  ▶ Generation or conversion of cryptographic material (keys, parameters)
  ▶ Management and storage of cryptographic objects (keys, certificates)
  ▶ Decouple client code from algorithm implementation
  ▶ Static factory method getInstance()
  ▶ Request implementation indirectly

```
static EngineClassName getInstance(String algorithm)
    throws NoSuchAlgorithmException
static EngineClassName getInstance(String algorithm, String provider)
    throws NoSuchAlgorithmException, NoSuchProviderException
static EngineClassName getInstance(String algorithm, Provider provider)
    throws NoSuchAlgorithmException
```
Message Digest

- Hash function

```java
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] data = getMessage();
byte[] hash = md.digest(data);
```

- Data provided in chunks using `update()` then call `digest()`

- If data is short and fixed - hashed in one step using `digest()`
Digital signature algorithms based on asymmetric encryption

Algorithm name: `<digest>with<encryption>`

**Sign:**

```java
byte[] data = "message to be signed".getBytes("ASCII");
Signature s = Signature.getInstance("SHA256withRSA");
s.initSign(privKey);
sig.update(data);
byte[] signature = sig.sign();
```

**Verify:**

```java
Signature s = Signature.getInstance("SHA256withRSA");
s.initVerify(pubKey);
s.update(data);
boolean valid = s.verify(signature);
```
Encryption and decryption operations

Encryption:

```java
Secret key = getSecretKey();

Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");

byte[] iv = new byte[c.getBlockSize()];
SecureRandom sr = new SecureRandom();
sr.nextBytes(iv);
IvParameterSpec ivp = new IvParameterSpec(iv);
c.init(Cipher.ENCRYPT_MODE, key, ivp);

byte[] data = "Message to encrypt".getBytes("UTF-8");
byte[] ciphertext = c.doFinal(data);
```
Decryption:

```java
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
c.init(Cipher.DECRYPT_MODE, key, ivp);
byte[] data = c.doFinal(ciphertext);
```
Message Authentication Code algorithms

```java
SecretKey key = getSecretKey();
Mac m = Mac.getInstance("HmacSha256");
m.init(key);
byte[] data = "Message".getBytes("UTF-8");
byte[] hmac = m.doFinal(data);
```
KeyGenerator

- Generates symmetric keys
- Additional checks for weak keys
- Set key parity when necessary
- Takes advantage of the cryptographic hardware

```java
KeyGenerator kg = KeyGenerator.getInstance("HmacSha256");
SecretKey key = kg.generateKey();
```

```java
KeyGenerator kg = KeyGenerator.getInstance("AES");
kg.init(256);
SecretKey key = kg.generateKey();
```
KeyPairGenerator

Generates public and private keys

```java
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024);
KeyPair pair = kpg.generateKeyPair();
PrivateKey priv = pair.getPrivate();
PublicKey pub = pair.getPublic();
```
Harmony’s Crypto Provider
- Limited JCA provider part of the Java runtime library
- SecureRandom (SHA1PRNG), KeyFactory (DSA)
- MessageDigest (SHA-1), Signature (SHA1withDSA)

Android’s Bouncy Castle Provider
- Full-featured JCA provider
- Part of the Bouncy Castle Crypto API
- Cipher, KeyGenerator, Mac, MessageDigest, SecretKeyFactory, Signature, CertificateFactory
- Large number of algorithms

Android OpenSSL Provider
- Native code, performance reasons
- Covers most functionality of Bouncy Castle
- Preferred provider
- Implementation uses JNI to access OpenSSL’s native code
Outline

Android Permissions

Cryptographic Providers

Network Security

Bibliography

Keywords
Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

- SSL is the predecessor of TLS
- Secure point-to-point communication protocols
- Authentication, Message confidentiality and integrity for communication over TCP/IP
- Combination of symmetric and asymmetric encryption for confidentiality and integrity
- Public key certificates for authentication
- Java Secure Socket Extension (JSSE)
Authentication

- Based on public key cryptography and certificates
- Both ends presents its certificate
- If trusted, they negotiate a shared key for securing the communication using pairs of public/private keys
- JSSE delegates trust decisions to TrustManager and authentication key selection to KeyManager
- Each SSLSocket has access to them through SSLContext
- TrustManager has a set of trusted CA certificates (trust anchors)
Obtain Trust Anchors

- Default JSSE TrustManager initialized using the system trust store
  - /system/etc/security/cacerts.bks

```java
TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init((KeyStore) null);

X509TrustManager xtm = (X509TrustManager) tmf.getTrustManagers()[0];

for (X509Certificate cert : xtm.getAcceptedIssuers()) {
    String certStr = "S:" + cert.getSubjectDN().getName() + "\nI:" + cert.getIssuerDN().getName();
    Log.d(TAG, certStr);
}
```
Use your own Trust Store

- Generate your trust store using Bouncy Castle and OpenSSL in command line
- Preferred HTTPS API

```java
KeyStore localTrustStore = KeyStore.getInstance("BKS");
InputStream in = getResources().openRawResource(R.raw.mytruststore);
localTrustStore.load(in, TRUSTSTORE_PASSWORD.toCharArray());

TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
 tmf.init(trustStore);

SSLContext sslCtx = SSLContext.getInstance("TLS");
sslCtx.init(null, tmf.getTrustManagers(), null);

URL url = new URL("https://myserver.com");
HttpsURLConnection urlConnection = (HttpsURLConnection) url
urlConnection.setSSLSocketFactory(sslCtx.getSocketFactory());
```
Android Security Internals, Nikolay Elenkov


https://github.com/nelenkov/custom-cert-https
Keywords

- Permissions
- Protection levels
- Static enforcement
- Dynamic enforcement
- Custom permissions

- Java Cryptography Architecture
- Cryptographic Service Provider
- Engine classes
- Java Secure Socket Extension
- Trust Store