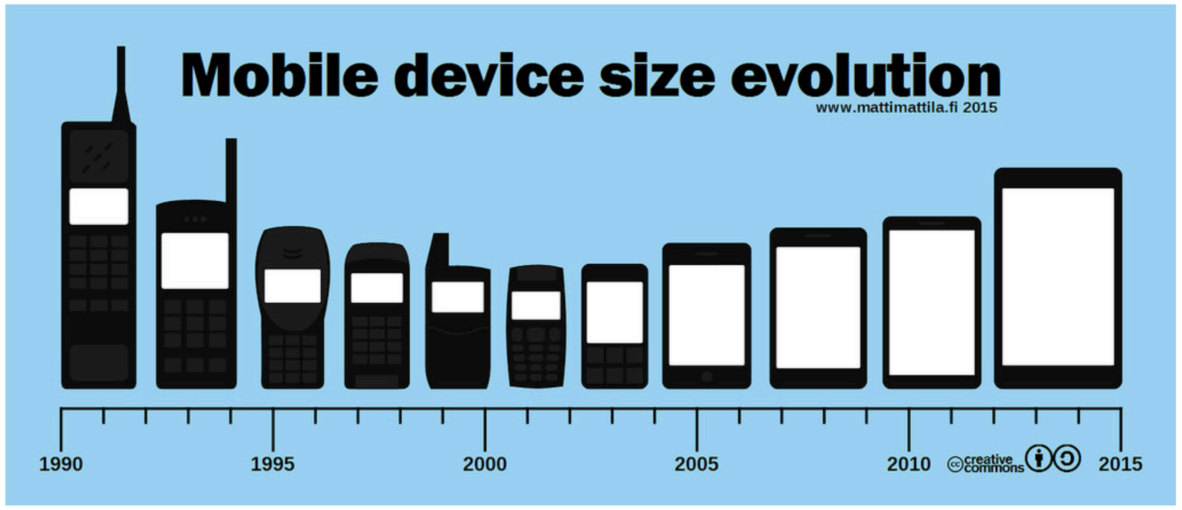
Mobile Platform Security

Paweł Szałachowski

Mobile Computing

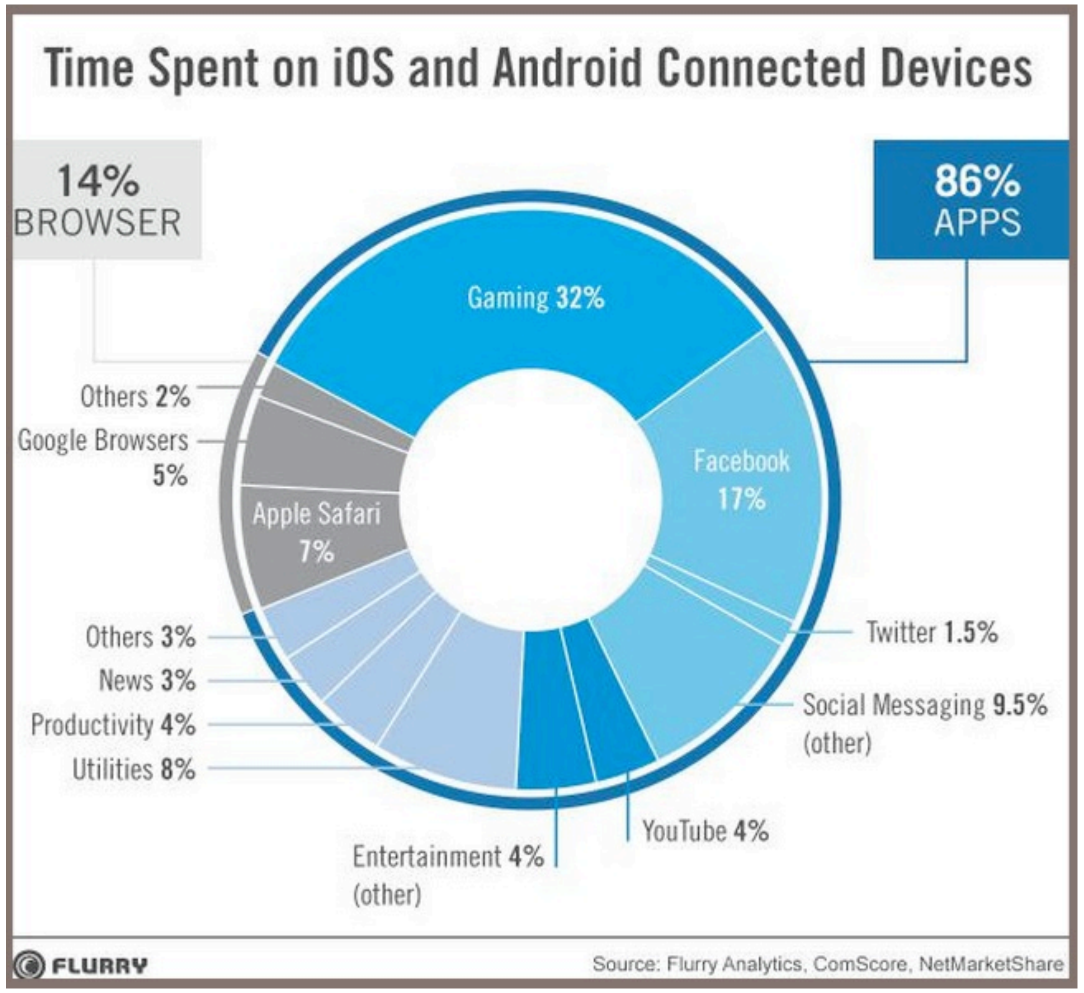
- Small devices we carry every day



- New well-integrated applications
- New vulnerabilities
- What is different (from the security point of view)?
  - Opportunity to create a secure architecture and infrastructure

Mobile Computing

- Mainframe -> Desktop/Server -> Mobile/Cloud
- Reliance on Cloud
- Security & Privacy more important
    - Could be designed from scratch

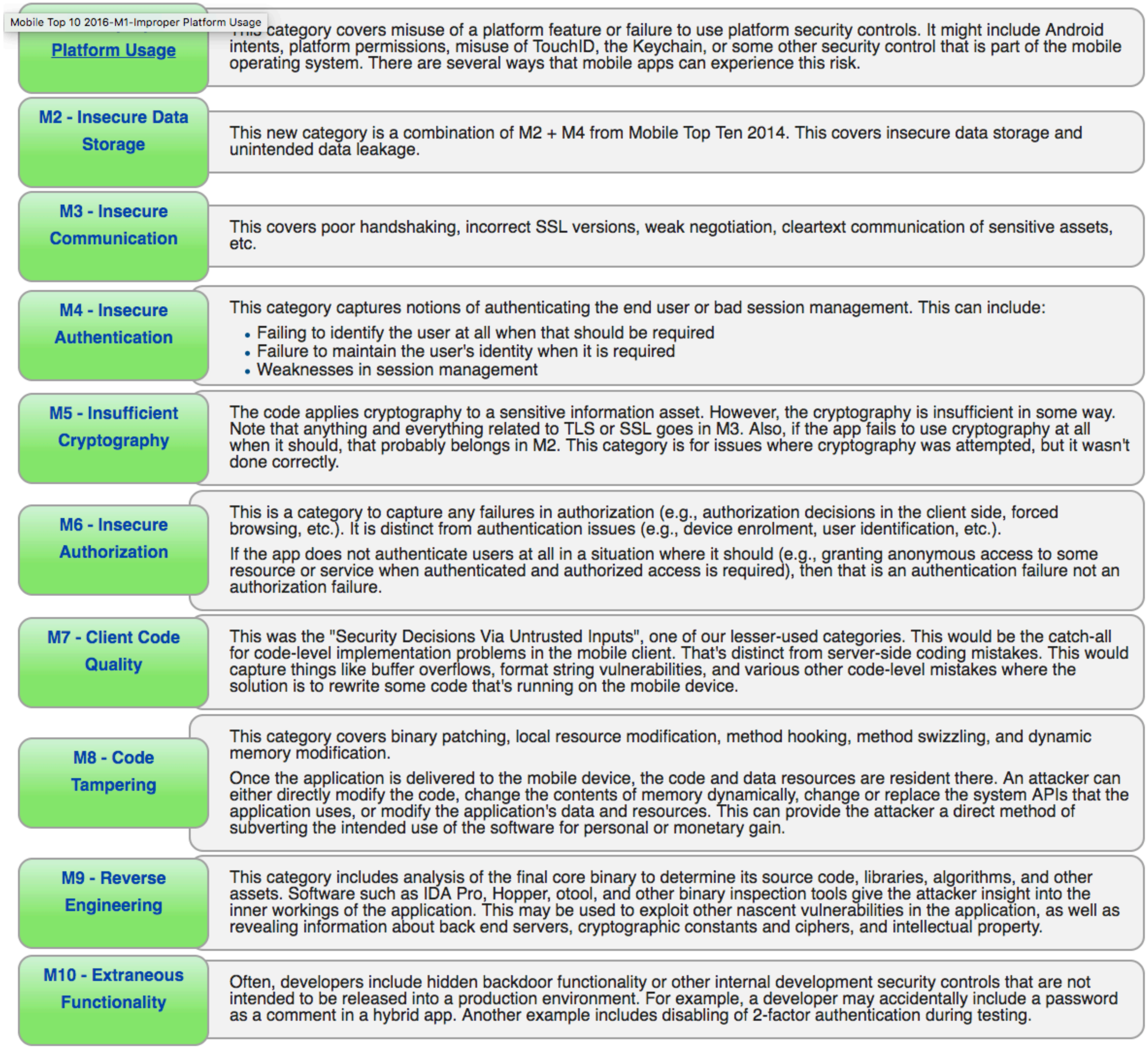


Apps

- App Marketplace
    - New model of software distribution
- iOS
    - Manual and automated verification
- Android
    - Easier to publish app
    - Removal through Bouncer
- App security
    - Isolation and protection
    - Sandboxing and permissions

Threat Model

- What is on your phone?
  - contacts, email, facebook, media (pics, video, music, …), location info and history, credentials (2FA, cloud/service access), …
- Threat models
  - Adversary with physical access
    - Try to break locking mechanism(s)
  - System-level attacks
    - Try to exploit vulnerabilities in mobile platform (software bugs, malformed data, …)
  - App attacks
    - Try to steal data or attack other apps by malicious app



Mobile Top 10 2016-M1-Improper Platform Usage

Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.
M2 - Insecure Data Storage	This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.
M3 - Insecure Communication	This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.
M4 - Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: • Failing to identify the user at all when that should be required • Failure to maintain the user's identity when it is required • Weaknesses in session management
M5 - Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for issues where cryptography was attempted, but it wasn't done correctly.
M6 - Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.
M7 - Client Code Quality	This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.
M8 - Code Tampering	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.
M9 - Reverse Engineering	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.
M10 - Extraneous Functionality	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.

Symantec Re

54%

More than half of organizations suffer mobile attacks

Posted on Marc

NIST

devi

Apr 2, 2

CATEGORIE

Physical, s
spectrum
elevators,
and moni
Standards
Standardi
Standardi

G+ Tweet

obile

ding the
g facilities,
ersal control
nal Institute of
bersecurity

BRIEF

NYC announces cybersecurity initiative to protect mobile devices

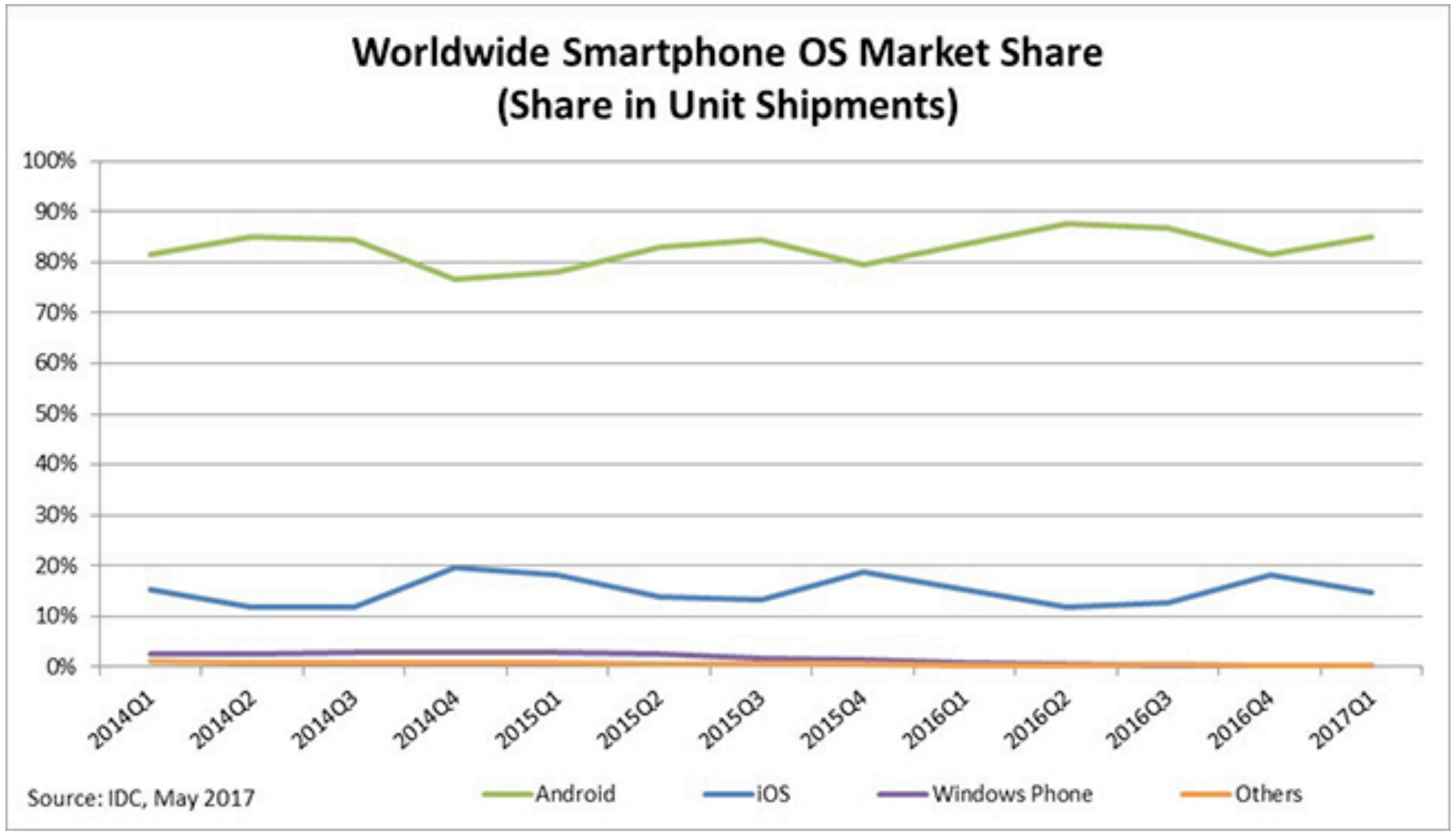
Physical Security

- Pin or pattern
  - Unlock all apps vs pin or pattern per app ?
  - Low entropy & smudge
- Fingerprint (with backup PIN)
  - No more secure than PIN
- Face unlock
  - Security concerns
- Tokens

Android

Android

- Late 2007: Android SDK
- Synchronized with Google Services
- Apps run on top of a Java middleware
- Modified Linux kernel
- Open-source
- Multiple vendors
- Marketplace for apps



Exploit Prevention

- Open source
  - Public audit (Does it work?)
- Goals
  - Prevent remote attacks and privilege escalation
- Overflow protection
  - ProPolice (stack protection), heap overflow protection, ASLR
- Sandbox (each app runs within an isolated VM)
- Apps announce permissions required
- Inter-apps communication is monitored

Android Apps

- There is no `main()` (nor its equivalent)
- Components
- Android Package (APK)
  - Manifest
    - Specifying components and defining policies
  - Binary code
  - App certificate and signatures

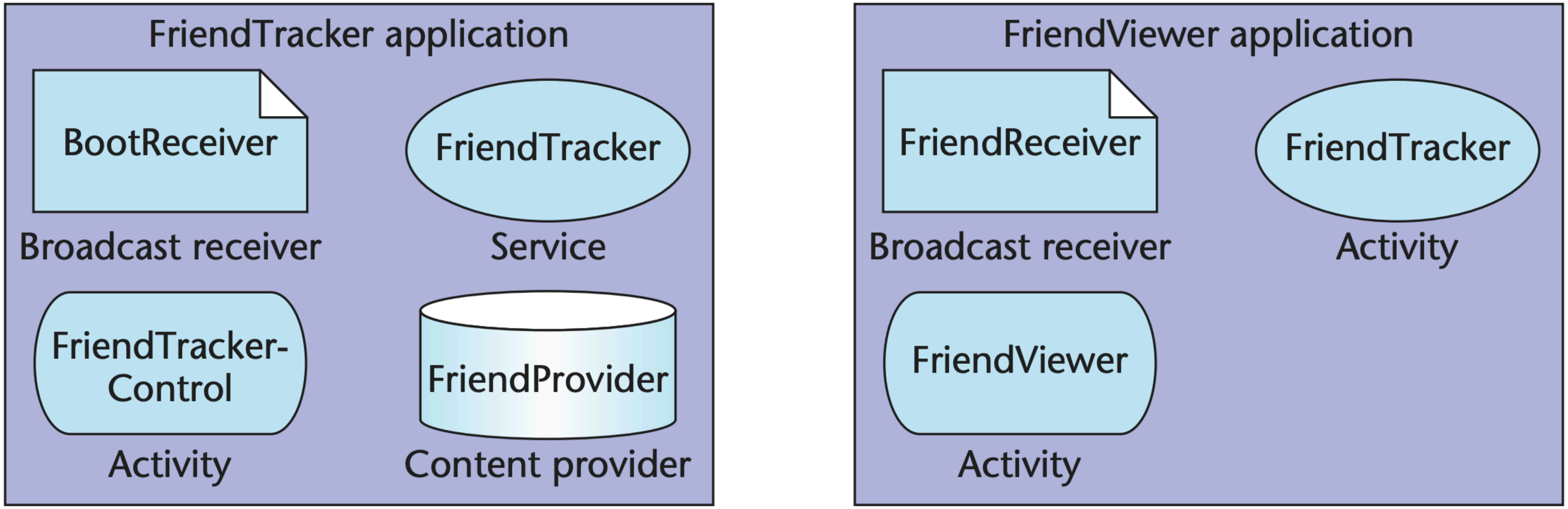
Component Types

- **Activity**: defines an app's user interface
  - Usually, one activity per *screen*
  - Activities start each other, can pass and return values
  - Only one activity has keyboard and processing focus at a time (other activities are suspended)
- **Service**: performs background processing
  - Operations that must continue after an activity is suspended
    - e.g., music, file download, or application-specific daemons starting on boot time
  - Often provide Remote Procedure Call (RPC)
- **Content provider**: stores and shares data
  - **Authority**: describes the content of a content provider
  - Other components can use the authority name to perform SQL queries
- **Broadcast receiver**: receives messages from other apps
  - Broadcast can be to implicit or explicit (!) destinations

Example App

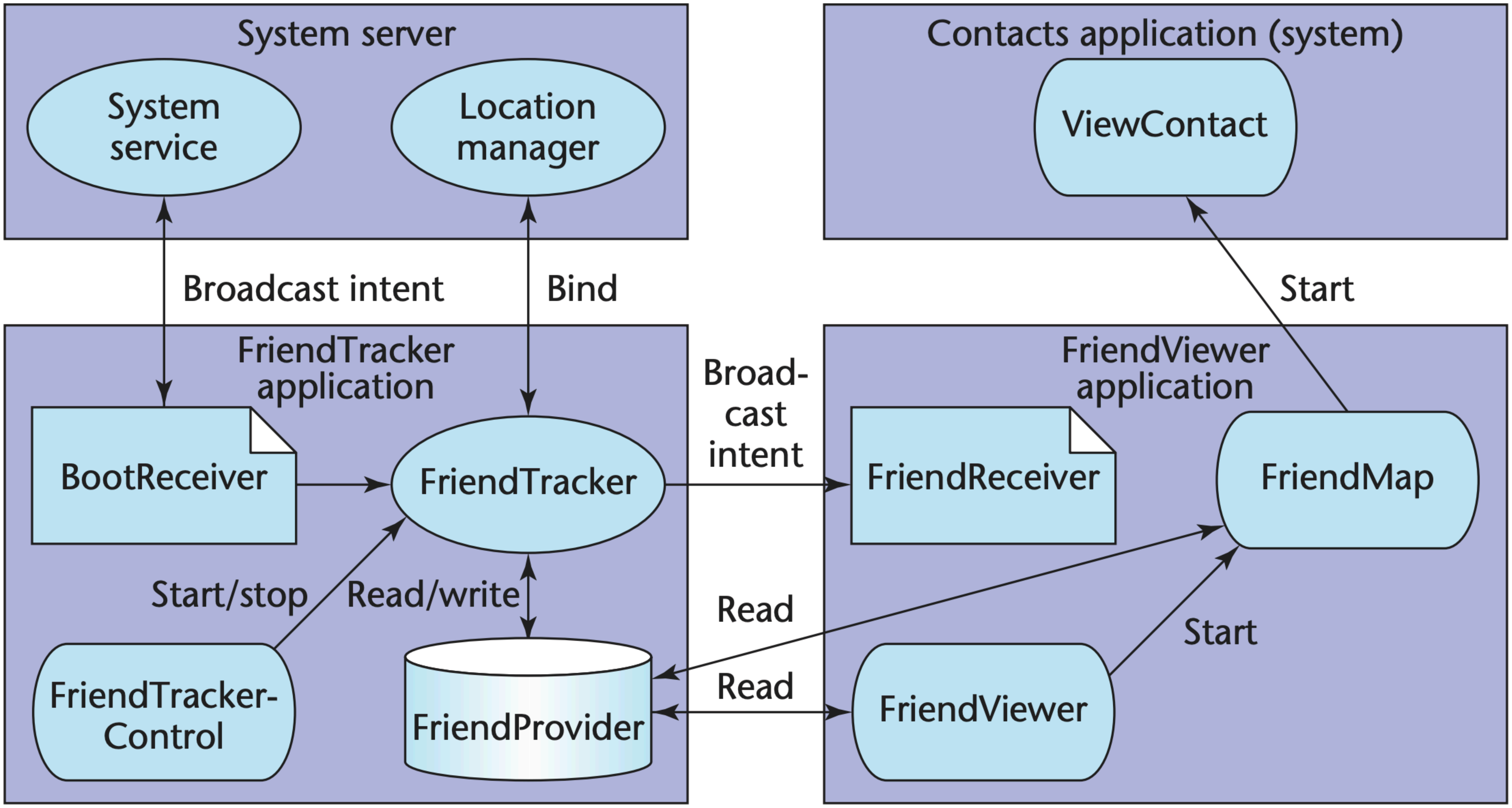
- Location-sensitive social networking app where users can discover their friends' locations
- **FriendTracker**: components for tracking, storing, and sharing friend locations
    - FriendTracker: polls an external service to discover friends' location
    - FriendProvider: maintains the most recent locations for friends
    - FriendTrackerControl: defines a user interface (start/stop tracking)
    - BootReceiver: receives a notification after system boots (used to start the service)
- **FriendViewer**: retrieves the stored locations and view friends on a map
    - FriendViewer: lists all friends and their locations
    - FriendMap: visualizes locations on a map
    - FriendReceiver: waits for messages indicating that the phone is near a friend (and displays it)

Example App



from Enck et al.

Example App



from Enck et al.

Inter-Component Communication (ICC)

- **Intent**: a message object (destination component address + data)
  - primary mechanisms for component interaction
- Android API defines methods for accepting intents which can trigger activities, services, or broadcast messages

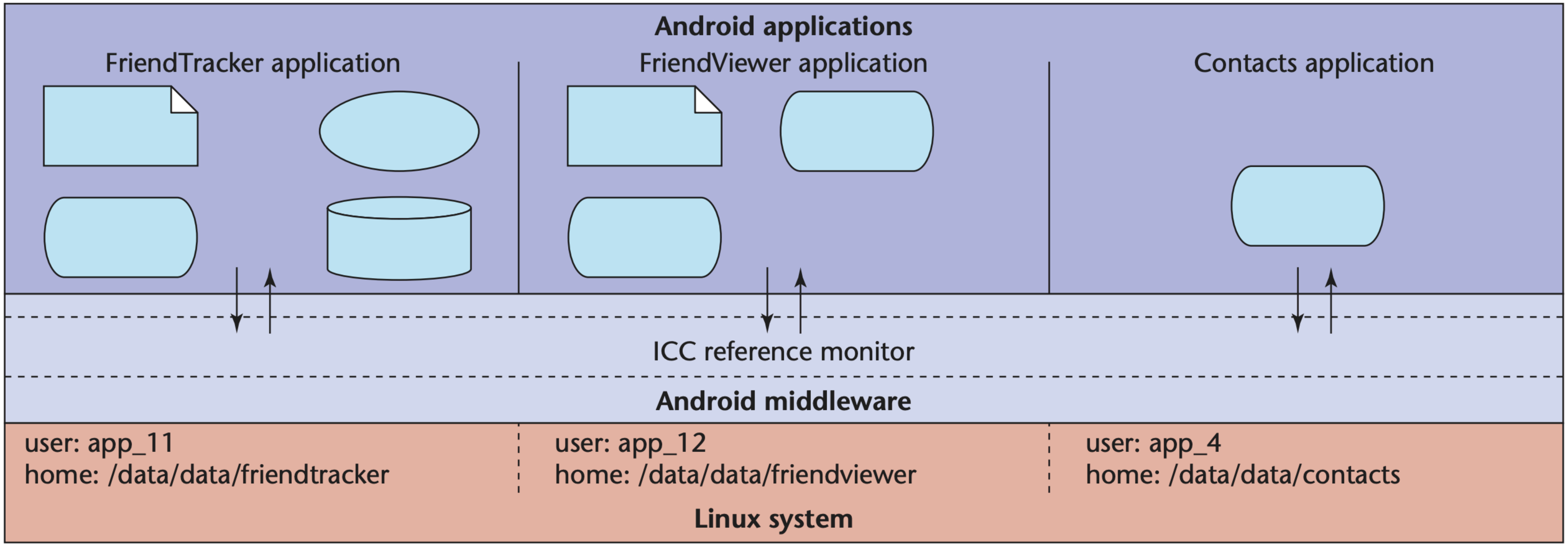
```
startActivity(Intent), startService(Intent), sendBroadcast(Intent)
```

- Destination can be uniquely addressed or an implicit name (**action string**)
  - Group of broadcast receivers can be addressed
  - Receivers set intent filters to limit action strings

Security Enforcement

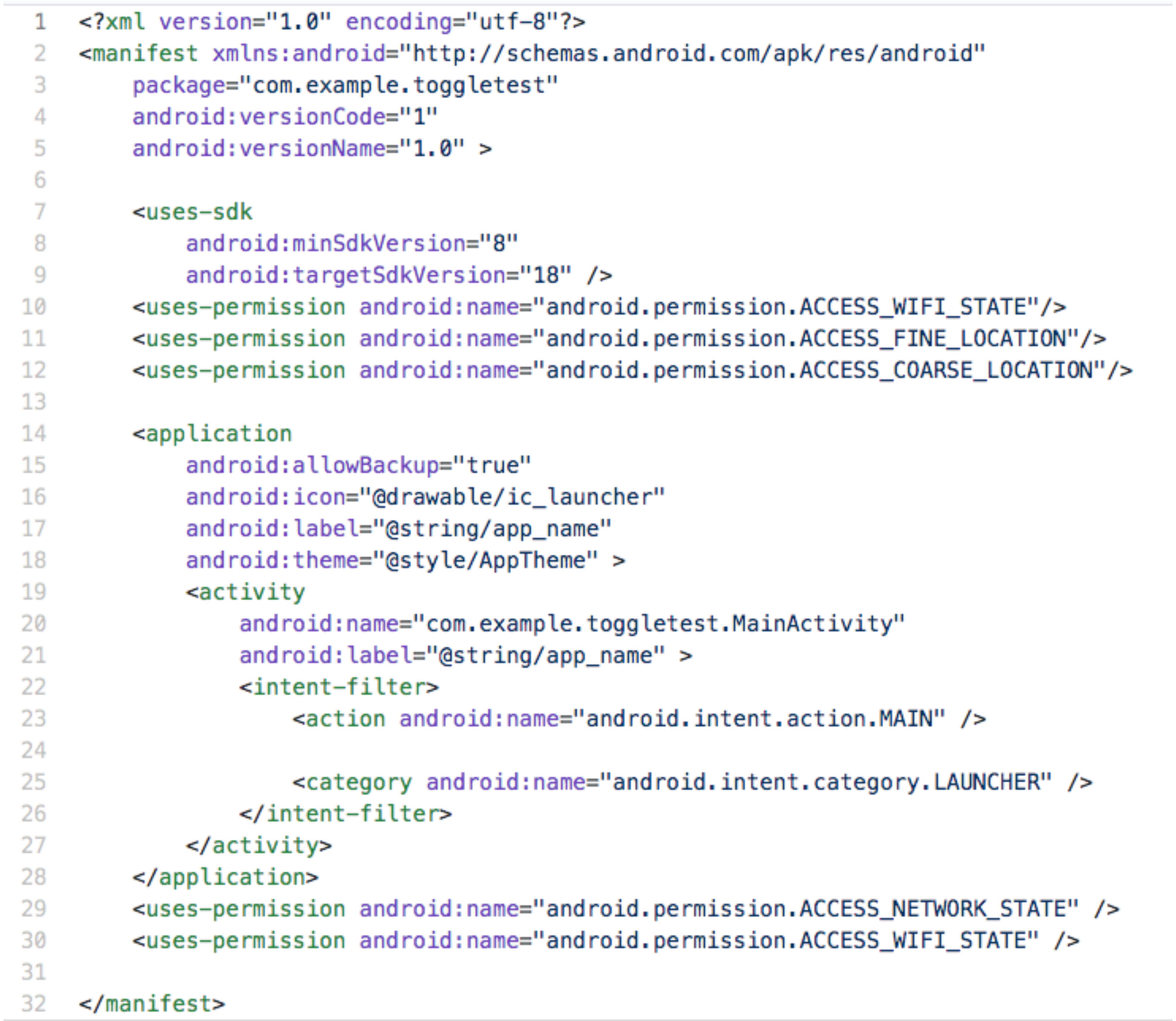
- Protection is implemented via two enforcement mechanisms
  - ICC-level and system-level enforcement
- System-level (Linux)
  - Each app runs as a unique user
    - Isolation & boundaries (memory, limits, home dir, …)
- ICC-level
  - ICC is not limited by user and process boundaries (cannot be)
    - `/dev/binder` device for I/O control
  - All ICC is based on apps' and components' **permission labels**
  - ICC reference monitor provides mandator access control (MAC)
    - Permissions (security policies) are placed in manifests defined by developers

Security Enforcement



from Enck et al.

Example Manifest



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.toggletest"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.toggletest.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

</manifest>
```

Security Refinements

Security Refinements

- Requirements
  - Basic security model is too basic
  - More security features needed
  - They can be considered as exceptions
- Some refinements introduce side effects
  - Implications may be difficult to understand

Private Components

- Apps contain components that others apps shouldn’t access (e.g., password- or key-related)
- Components can be private (accessed only by components from the same app)

  ```
  exported=False
  ```

  - No permission assignment needed
- By default, components are public

Implicitly Open Components

- Intent filters on activities are popular
    - Developers set them
- Callers do not know (beforehand) what access permission is required
- Developers of target activities can declare it open
    - public component w/o access permission can be accessed by any application
- Should be applied only exceptionally
    - Easy to use vs secure

Broadcast Intent Permissions

- Broadcast intent can be read by all applications, that can lead to data leaks or privacy issues (e.g., our contact is close to us)

- Developers can limit set of receiving apps

  - `sendBroadcast(intent, “perm.PERM1”)` will send it only to apps containing the “`perm.PERM1`” permission label

- Issues?

  - It is not part of manifest files

Content Provider Permissions

- Content providers provide interfaces for reading (SELECT) or writing (INSERT, UPDATE, and DELETE) data
  - Often our app should be the only one to write, while other apps can read
- Developers can assign both read and write permissions (instead of using one permission label)

Service Hooks

- A component with the permission can start, stop, or bind the service
  - More flexible permissions needed
    - e.g., `isAlive`() is read-only, while `addEntry`() writes
- `checkPermission`() method to let developers define arbitrary permission checks
- Implemented at the code level

Protected APIs

- Some system resources (net, camera, mic) are accessed through special APIs (not as components)
- These APIs are additionally protected
  - App has to declare a corresponding permission labels (in manifest) to use them
    - e.g., `perm.INTERNET`

Permission Protection Levels

- Several permission protection levels to protect users’ privacy
- There are three protection levels that affect third-party apps
    - Normal: cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps
    - Signature: the system grants these app permissions at install time, but only when the app that attempts to use a permission is signed by the same certificate as the app that defines the permission
    - Dangerous: cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps

Pending Intents

- Delegate actions to other apps
  - e.g., app can pass a Pending Intent to other apps enabling them to invoke service on behalf of the requesting app
  - Implemented as a reference pointer
- It is deviation from the Android MAC model
  - Intents are delegated

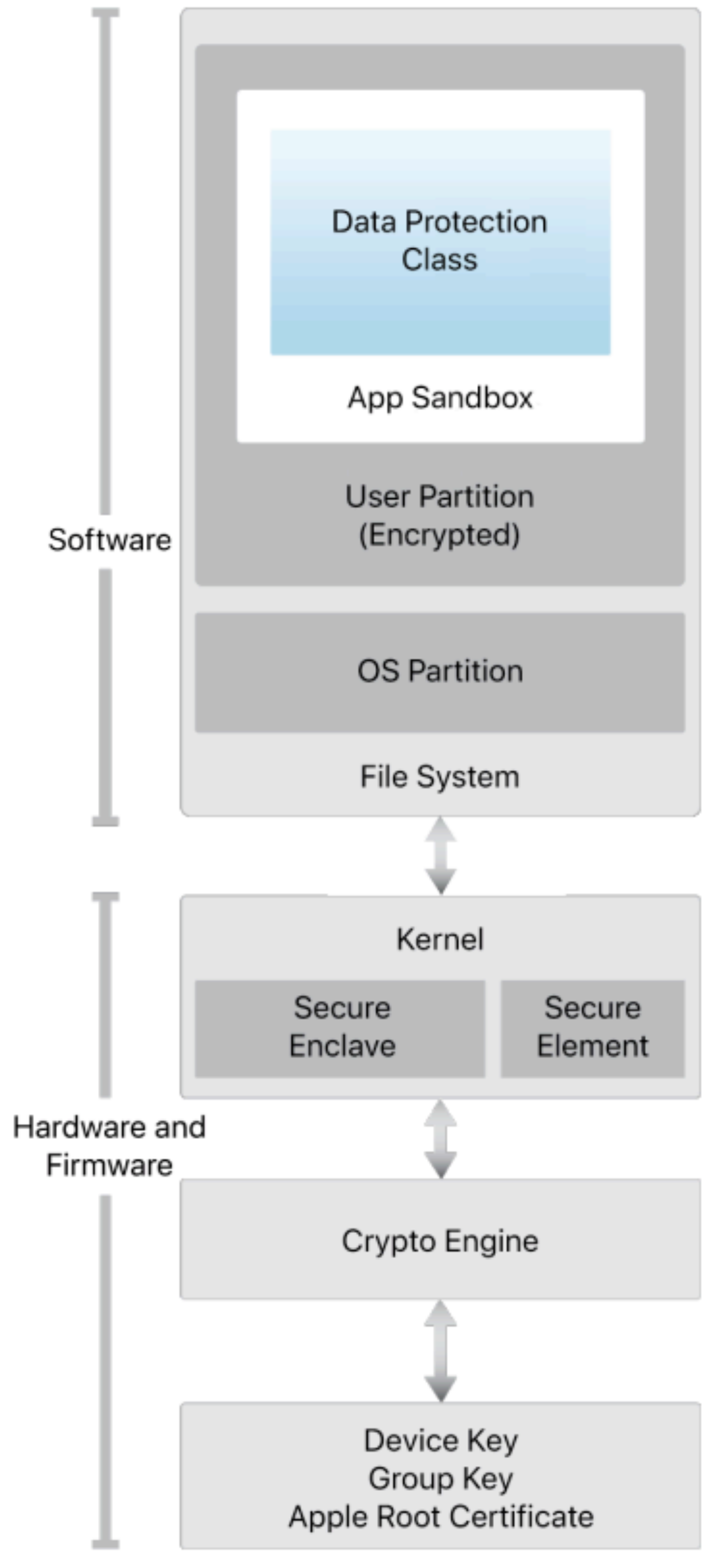
URI Permissions

- Permissions to content can be implemented via special content URIs
    - An app that has a URI permission does not have to have a read permission access to the content provider
    - e.g., e-mail app and viewer app (reading jpeg attachment)
- It is a delegation again (i.e., MAC is violated)

Apple iOS

iOS Security

- Data security
  - Prevent unauthorized use of device
  - Protect data at rest
- Network security
- App security

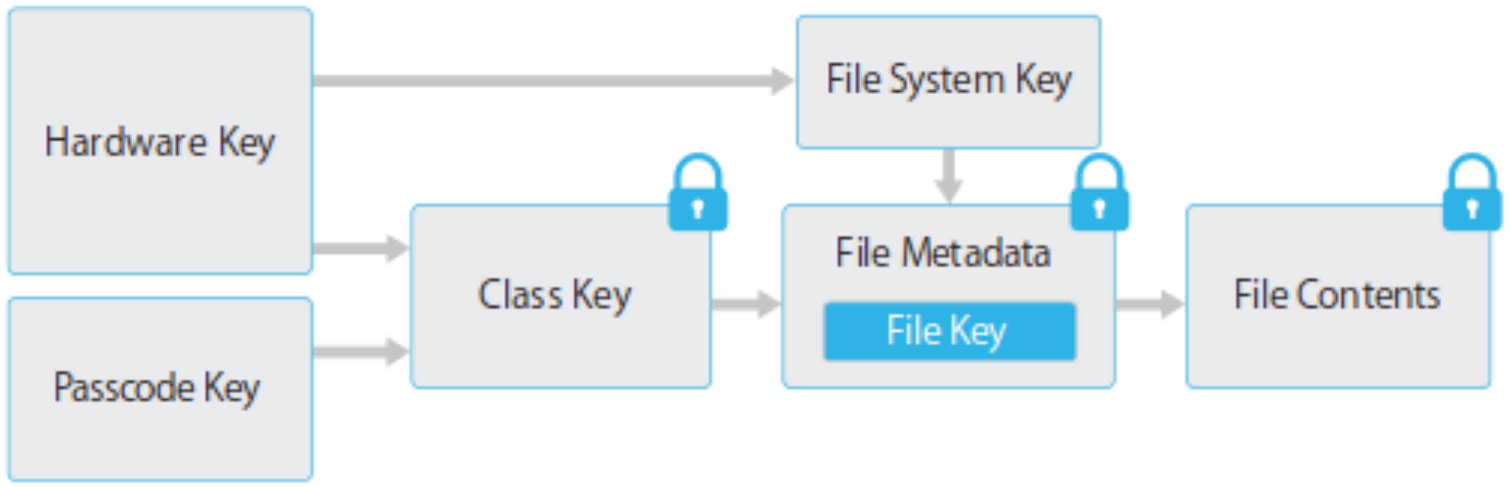


Secure Boot

- Every component ensures that the next one is signed properly
- Boot ROM is the root of trust
  - Installed during manufacturing
- Only authorized iOS code can boot
  - then, how does Jailbreaking work?

Software Security

- All iOS software updates are signed by Apple
  - Devices have unique ID tracked by Apple (which devices were updated)
- Data protection
  - Writes to flash are encrypted (AES-XTS)
  - Keys: per-file key, class key, file-system key
  - Enc/Dec within the secure enclave
  - Backup (iCloud)
- App exploit mitigation
  - NX and ASLR



App Security

- Runtime Protection
  - User apps / kernel space isolation
  - Apps are sandboxed
  - Inter-app communication through iOS APIs
- Mandatory code signing
  - Using Apple-issued certificate
- Apps can use hardware encryption

Reading

- https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/android.pdf
- https://www.guanotronic.com/~serge/papers/soups12-android.pdf
- https://www.apple.com/business/docs/iOS_Security_Guide.pdf
- https://source.android.com/security/

Questions ?