CS365 Presentation Notes

**System and Kernel security**

1. Android platform provides process level security by the Linux kernel, as well as a secure inter-process communication (IPC) to enable secure communication between applications running in different processes.
2. These security features ensure that even native code at the OS level runs in the Application Sandbox.
3. This isolates a process to itself and won’t let it harm any other processes, the kernel, or OS level components.

**Linux Security**

1. The Linux kernel is at the base of the Android platform.
2. The Linux kernel has been in widespread use for years, and is used in millions of security-sensitive environments. Through its history of constantly being researched, attacked, and fixed by thousands of developers, Linux has become a stable and secure kernel trusted by many corporations and security professionals.

**As the base for Android, the Linux kernel provides it with several key security features, including:**

1. **A user-based permissions model**
   1. Meaning every user gets distinct permissions not like Linux where a many applications run with the same user permissions.
2. **Process isolation**
   1. Meaning a process is shielded from outside elements/processes
3. **Extensible mechanism for secure IPC**
   1. Processes can talk to each other in a secure manor.
4. **The ability to remove unnecessary and potentially insecure parts of the kernel**.
   1. Meaning they write their own code for the kernel and remove anything they don’t need.

**As a multiuser operating system, a fundamental security objective of the Linux kernel is to isolate user resources from one another. Thus, Linux:**

1. Prevents user A from reading user B's files
2. Ensures that user A does not exhaust user B's memory
3. Ensures that user A does not exhaust user B's CPU resources
4. Ensures that user A does not exhaust user B's devices (e.g. telephony, GPS, Bluetooth)

**The Application Sandbox**

1. Android takes advantage of the Linux user-based protection as a means of identifying and isolating application resources. The Android system assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process.
2. This sets up a kernel-level Application Sandbox. The kernel enforces security between applications and the system at the process level using user and group IDs that are assigned to applications.
3. By default, applications cannot interact with each other and applications have limited access to the operating system. If application A tries to do something malicious like read application B's data or dial the phone without permission, then the operating system protects against this because application A does not have the appropriate user privileges.
4. Because the Application Sandbox is in the kernel, this security model extends to native code and to operating system applications.
5. All of the software above the kernel, such as operating system libraries, application framework, application runtime, and all applications, run within the Application.
6. Because all applications and their resources are sandboxed at the OS level, a memory corruption error will allow code execution only in the context of that particular application, with the permissions established by the operating system.

**Rooting of Devices**

1. On Android, only the kernel and a small subset of the core applications run with root permissions.
2. Android does not prevent a user or application with root permissions from modifying the operating system, kernel, or any other application.
3. In general, root has full access to all applications and all application data. The ability to modify an Android is important to developers working with the Android platform.
4. On many Android devices users have the ability to unlock the bootloader in order to allow installation of alternate operating systems.
5. These alternate operating systems allow a developer to gain root access for debugging applications and system components or to access features not presented to applications by Android APIs.

**Hardware Abstraction Layer (HAL)**

1. The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.
2. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth modules.

**Android Runtime**

1. For devices running Android version 5.0 or higher, each app runs in its own process and with its own instance of the Android Runtime (ART).
2. ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed especially for Android that's optimized for minimal memory footprint.
3. Build toolchains, such as Jack, compile Java sources into DEX bytecode, which can run on the Android platform.

**Some of the major features of ART include the following:**

* 1. **Ahead-of-time (AOT) and just-in-time (JIT) compilation**
  2. **Optimized garbage collection** (GC)
  3. **Better debugging support**, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set breakpoints to monitor specific fields

1. Prior to Android version 5.0, “Dalvik” was the Android runtime. If your app runs well on ART, then it should work on Dalvik as well, but the reverse may not be true.

**Native C/C++ Libraries**

1. Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++.
2. The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps.
   1. For example, you can access OpenGL ES through the Android framework’s Java OpenGL API to add support for drawing and manipulating 2D and 3D graphics in your app.
3. If you are developing an app that requires C or C++ code, you can use the Android NDK to access some of these native platform libraries directly from your native code.

**Java API Framework**

1. The entire feature-set of the Android OS is available through APIs written in the Java language.
2. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:
   1. A View System you can use to build an app’s UI.
   2. A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
   3. A Notification Manager that enables all apps to display custom alerts in the status bar
   4. An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack
   5. Content Providers that enable apps to access data from other apps, such as the Contacts app, SMS app, Telephony app, or to share their own data
3. Developers have full access to the same framework APIs that Android system apps use.

**System Apps**

1. Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more.
2. Apps included with the platform have no special status among the apps the user chooses to install. So a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions apply, such as the system's Settings app).
3. The system apps function both as apps for users and to provide key capabilities that developers can access from their own app.
   1. For example, if your app would like to deliver an SMS message, you don't need to build that functionality yourself—you can instead invoke whichever SMS app is already installed to deliver a message to the recipient you specify.