**2.1.2 Anti-debugging techniques**

JDWP is a protocol for communication between the debugger and the Java virtual machine (VM) that it debugs. JDWP is a standard debugging protocol that's supported by all command line tools and Java IDEs, including JDB, JEB, IntelliJ, and Eclipse, A JDWP debugger allows you to step through Java code, set breakpoints on Java methods, and inspect and modify local and instance variables. JDWP debugger used most of the time when debug "normal" Android apps.

The Android application package file, APK file, can be easily decompiled using Android reverse engineering tools. Thus, general apps can be easily transformed into malicious application through reverse engineering and analysis. These repacked apps could be uploaded in general android app market. To prevent theses malicious behaviors such as malicious code injection or code falsifications, many techniques and tools were developed. However, these techniques also can be analyzed using debuggers. Also, analyzed apps can be tampered easily. For example, when applying anti-analysis techniques to android apps using DexGuard, it can be seen that these techniques can also be analyzed using debugger. so, to protect the android app from the attack using debugger, we propose anti-debugging techniques for code and managed code debugging of android apps. [3]

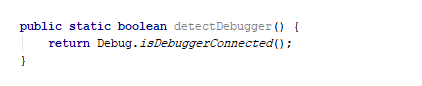
1. **Checking Debuggable Flag in ApplicationInfo**

when JDWP thread start for app, the value of flag "android: debuggable" in manifest determines, its value can be determined programmatically using the app's ApplicationInfo object. If the flag is set then debug enable, as shown in code below:



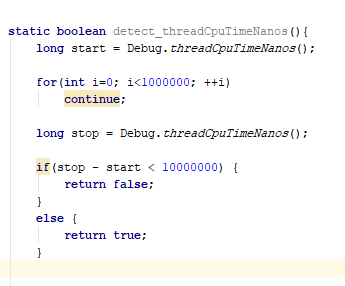
1. **static method**

The Android Debug system class offers a static method for checking whether a debugger is currently connected, we can use a class as shown below:



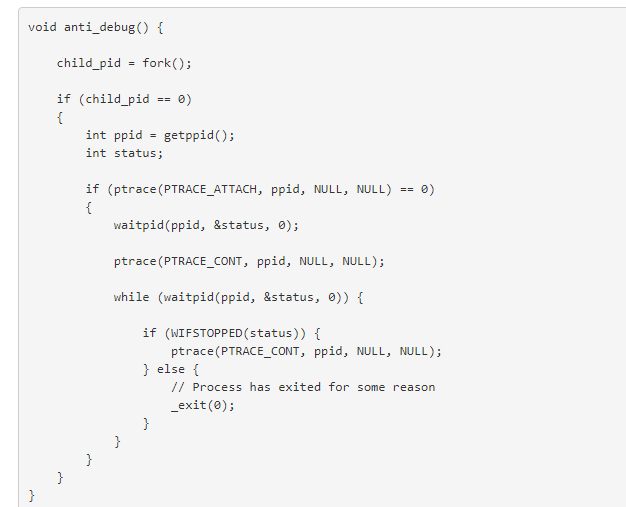
1. **Timer Checks**

The "Debug.threadCpuTimeNanos" indicates the amount of time that the current thread has spent executing code. As debugging slows down execution of the process, [the difference in execution time can be used to make an educated guess on whether a debugger is attached](https://slides.night-labs.de/AndroidREnDefenses201305.pdf).



1. **Checking TracerPid**

On Linux, The **"ptrace**()" system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers, A straightforward way of using the ptrace system call for anti-debugging is forking a single child, and then calling *ptrace(parent\_pid)* to attach to the parent.[4]



If implemented as above, the child will keep tracing the parent process until the parent exits, causing future attempts to attach a debugger to the parent to fail. We can verify this by compiling the code into a JNI function and packing it into an app we run on the device.

**References:**

3.Anti-debugging scheme for protecting mobile apps on android platform <https://link.springer.com/article/10.1007/s11227-015-1559-9/> Retrieved Nov.2017

43 Android Anti-Debugging Fun <http://www.vantagepoint.sg/blog/89-more-android-anti-debugging-fun/> Retrieved Nov.2017