Going Native: Using a Large-Scale Analysis of Android Apps to Create a Practical Native-Code Sandboxing Policy

MAR 4TH, 2016

论文下载

摘要

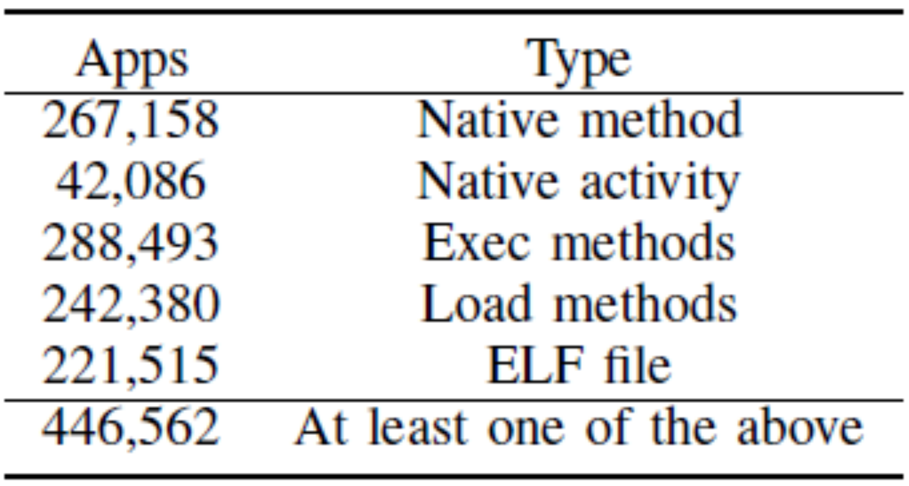
当前的静态分析工具更多关注的是java层的分析，而忽略了对于native上的分析。少部分针对native的保护工具更多的是一味的关心如何减少native的权限，而没有现实app的佐证。所以作者首先静态分析了1,208,476个app，找到其中使用native的app（446,562）。然后通过动态分析的方式分析这些native code的行为，然后总结出一个针对native的保护策略。

分析流程

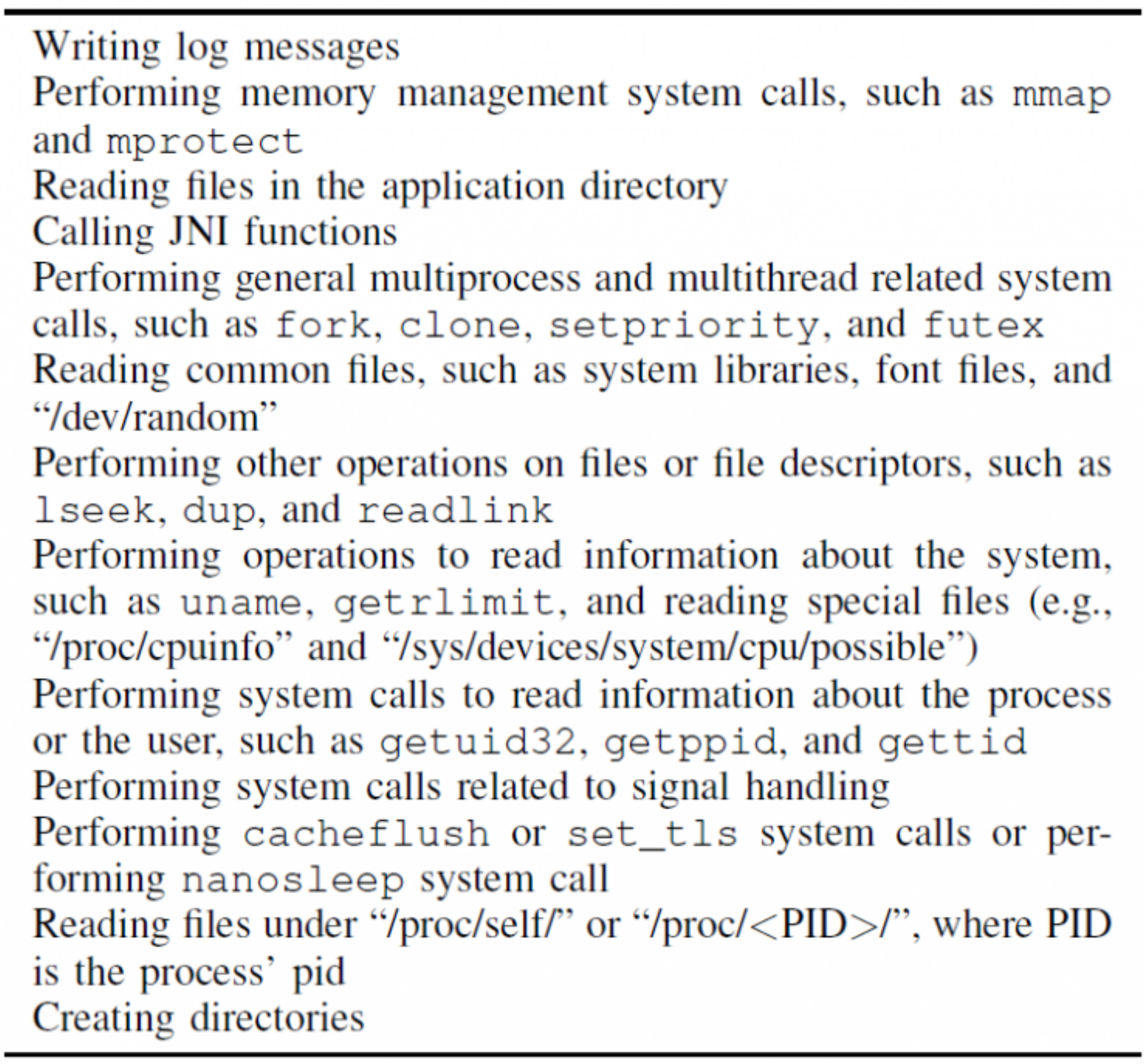
- 利用Androguard作为基本的工具，从1,208,476个APP中挑选使用native code的app。
- 作者自己做了个安卓模拟器，用来监控app中native code的行为。并且利用Monkey、恶意的intent的方式模拟app的输入。
- 根据动态分析的native的action，列一个白名单，当有99%的app有相似的行为就认为是正常行为。

实验

TABLE I. RESULTS OF THE STATIC ANALYSIS.

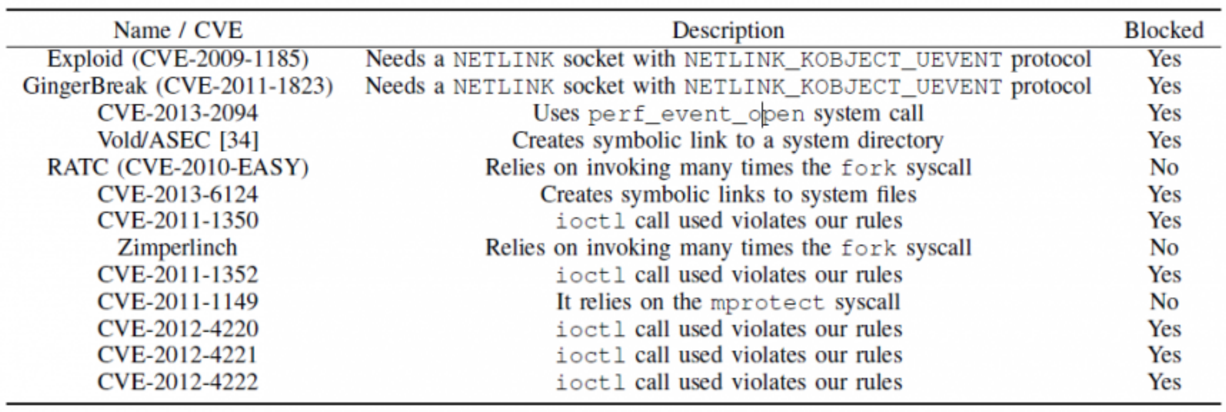


Apps	Type
267,158	Native method
42,086	Native activity
288,493	Exec methods
242,380	Load methods
221,515	ELF file
446,562	At least one of the above



Writing log messages
Performing memory management system calls, such as `mmap` and `mprotect`
Reading files in the application directory
Calling JNI functions
Performing general multiprocess and multithread related system calls, such as `fork`, `clone`, `setpriority`, and `futex`
Reading common files, such as system libraries, font files, and "/dev/random"
Performing other operations on files or file descriptors, such as `lseek`, `dup`, and `readlink`
Performing operations to read information about the system, such as `uname`, `getrlimit`, and reading special files (e.g., "/proc/cpuinfo" and "/sys/devices/system/cpu/possible")
Performing system calls to read information about the process or the user, such as `getuid32`, `getppid`, and `gettid`
Performing system calls related to signal handling
Performing `cacheflush` or `set_tls` system calls or performing `nanosleep` system call
Reading files under "/proc/self/" or "/proc/<PID>/", where PID is the process' pid
Creating directories

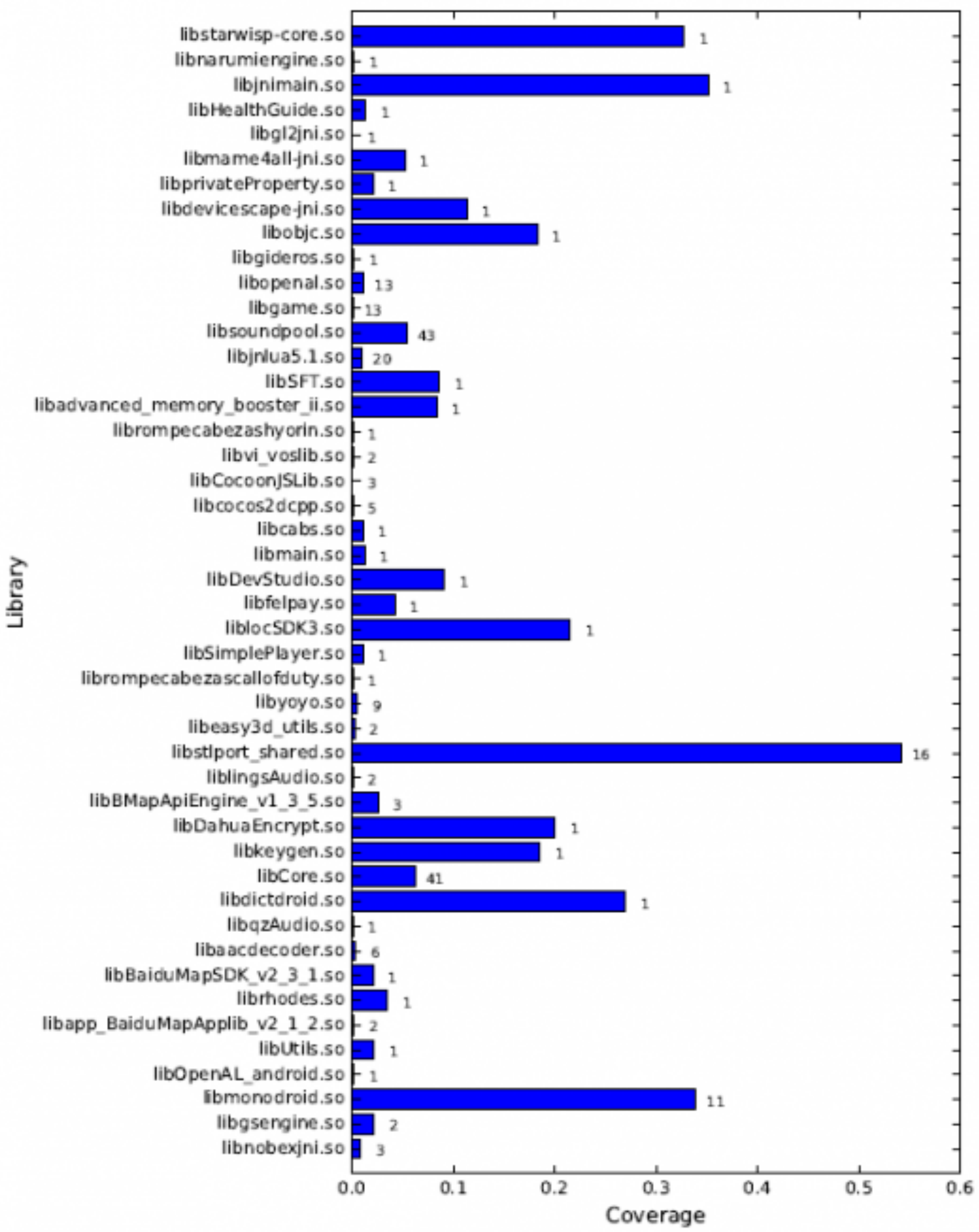
- 利用上述策略，阻止了1,414 apps (0.12%)
- 测试了13 root exploits，有10个被成功阻拦



Name / CVE	Description	Blocked
Exploid (CVE-2009-1185)	Needs a `NETLINK` socket with `NETLINK_KOBJECT_UEVENT` protocol	Yes
GingerBreak (CVE-2011-1823)	Needs a `NETLINK` socket with `NETLINK_KOBJECT_UEVENT` protocol	Yes
CVE-2013-2094	Uses `perf_event_open` system call	Yes
Vold/ASEC [34]	Creates symbolic link to a system directory	Yes
RATC (CVE-2010-EASY)	Relies on invoking many times the `fork` syscall	No
CVE-2013-6124	Creates symbolic links to system files	Yes
CVE-2011-1350	`ioctl` call used violates our rules	Yes
Zimperlinch	Relies on invoking many times the `fork` syscall	No
CVE-2011-1352	`ioctl` call used violates our rules	Yes
CVE-2011-1149	It relies on the `mprotect` syscall	No
CVE-2012-4220	`ioctl` call used violates our rules	Yes
CVE-2012-4221	`ioctl` call used violates our rules	Yes
CVE-2012-4222	`ioctl` call used violates our rules	Yes

动态分析的代码覆盖率

- 随机挑选了25,000，Java的覆盖率8.31%


Library
libstarwisp-core.so 1
libnarumiengine.so 1
libjnimain.so 1
libHealthGuide.so 1
libgl2jni.so 1
libmame4all-jni.so 1
libprivateProperty.so 1
libdevicescape-jni.so 1
libobjc.so 1
libgideros.so 1
libopenal.so 13
libgame.so 13
libsoundpool.so 43
libjnlua5.1.so 20
libSFT.so 1
libadvanced_memory_booster_ii.so 1
librompecabezashyorin.so 1
libvi_voslib.so 2
libCocoonJSLib.so 3
libcocos2dcpp.so 5
libcabs.so 1
libmain.so 1
libDevStudio.so 1
libfelpay.so 1
liblocSDK3.so 1
libSimplePlayer.so 1
librompecabezascallofduty.so 1
libyoyo.so 9
libeasy3d_utils.so 2
libstlport_shared.so 16
liblingsAudio.so 2
libBMapApiEngine_v1_3_5.so 3
libDahuaEncrypt.so 1
libkeygen.so 1
libCore.so 41
libdictdroid.so 1
libqzAudio.so 1
libaacdecoder.so 6
libBaiduMapSDK_v2_3_1.so 1
librhodes.so 1
libapp_BaiduMapApplib_v2_1_2.so 2
libUtils.so 1
libOpenAL_android.so 1
libmonodroid.so 11
libgsengine.so 2
libnobexjni.so 3
Coverage