The Genie in the Market

By Scott Dunlop, Senior Security Consultant at IOActive, Inc

Abstract: The Android Market is an open and friendly variation on the app stores spreading across the mobile phone industry. These applications appear safe on the surface, but they exact a price for developer accessibility that is paid by unsuspecting Android consumers and vendors. This article discusses the threats presented by native libraries included by Android Market applications and covers how these vulnerabilities were exploited by the Unrevoked app to jailbreak the latest generation of Android phones.

The Idealized Android Security Model

Application processes and information are isolated from each other in Android using Linux user and group identifiers. Each application is assigned its own UID and GID at installation, and has read-only access to the installed package. Persistent data maintained by the application is also kept private using these identifiers.

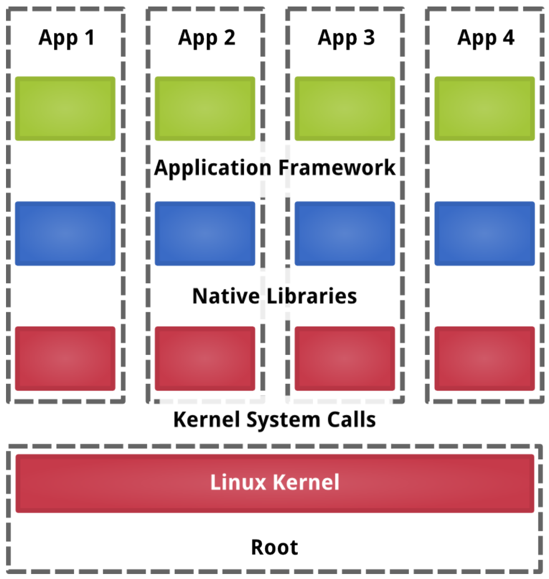

Figure 1. "The Android Security Ideal"

By default, applications in Android have very restricted access to the hardware and to shared data. Access to sensitive hardware and operating system services is generally provided by the introduction of supplemental group identifiers to the application, such as the "graphics" or "bluetooth" groups. (The UNIX model generally associated one effective user identifier and one group identifier with a process, and permits zero or more supplemental group identifiers to expand access.)

Because this type of access control model is fairly coarse, additional protections are layered on top of this variation of mandatory access control. The Android application frameworks will also restrict access to sensitive functions based on the capabilities required by the application.

For Java applications, this tiered approach works well. Applications that have not specified a requirement for a given capability simply cannot use portions of the framework that require that capability. This is particularly useful in limiting access by untrusted applications to more privileged applications, such as the browser or media players.

How the Market Leans on the Idealized Security Model

The Apple and Microsoft curated model for application stores has received considerable discussion. In this model, an application developer must submit the application for review before publication to the store for purchase by end-users. Google has taken a different route with the Android market; developers may publish their applications directly to the market.

End users are expected to make a well-informed decision about the potential risk presented by the application by assessing the security exceptions required by the application. (This article ignores the conditioning of end users to overlook cautionary statements due to overuse.)

The Genie in the Marketplace

The hardware targeted by Android and other mobile operating systems tends to be tightly constrained by form factor and power requirements. While major inroads have been made into the performance of managed code runtimes, such as Android's Dalvik virtual machine, the need frequently arises to improve performance by using unmanaged native code for demanding portions of the application.

The Android application model supports this approach by permitting the inclusion of Java Native Interface libraries in market applications. A developer can bypass the code safety controls imposed by Dalvik and Java, including range checking and exception handlers, by writing portions of the application in C or even machine code.

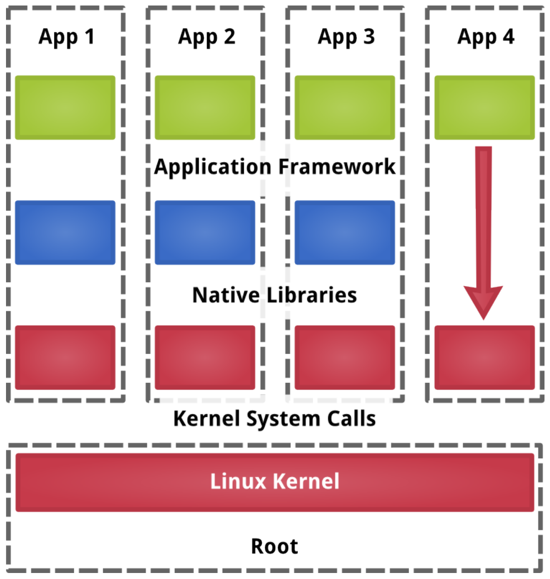

Figure 2. "The Native Bypass"

The Java Native Interface permits direct access to native code libraries and the kernel system call interface. As a result, the application can now bypass the security controls enforced by the

Android application frameworks. A malicious application developer can then exploit the much coarser access controls provided by the operating system or apply privilege escalation attacks against the kernel, exposed devices, and setuid utilities present on the device.

Even native code written with the purest intentions can lead to an increased threat surface because of the lack of protection against buffer overflows and other classic binary-level vulnerabilities in native applications. Due to the limitations of the underlying architecture, the common mitigations against binary exploitation such as ASLR and nonexecutable stacks may not be present, making the successful exploitation of a susceptible application more likely.

The Genie in the Wild—The Unrevoked Jailbreak

Soon after the release of Sprint's new Evo phone, running Android 2.1, a jailbreak was provided on the Android Marketplace by the "Unrevoked" team. This application exploited a native code portion of a bundled Sprint application with privilege levels that exceeded those commonly available on the Marketplace.

To do this, Unrevoked included a native code utility that it executed using the "Exec" method of the java.lang.Runtime class. This utility then used a yet-to-be-disclosed vulnerability to elevate privileges and install a "su" utility that provides the end user with a durable means of gaining root access to the device.

In the eyes of end users, Unrevoked is relatively benign. But the techniques demonstrated could just as easily have been used to install a rootkit module into the kernel that could exploit the trust of the user for the mobile device in an insidious, ongoing fashion similar to the malware epidemic currently plaguing personal computers on the desktop.

Putting the Genie Back in the Bottle

In the case of Unrevoked, Sprint distributed a bug fix to close the hole. Google has also demonstrated the ability to uninstall applications remotely, a practice that has recently raised concerns with consumer-rights advocates. Post-facto solutions, such as patching vulnerabilities and trying to uninstall malware, have proven to be a poor solution in the personal computer sector.

As long as direct access to native hardware is available to anonymous developers on the market, the operating system and underlying privilege model will be assaulted by malware authors who show increasing interested in mobile devices. Java Native Interface libraries should be permitted in the public market only if thoroughly code-reviewed and associated with an individual who has something to lose if caught producing malware.

Additionally, vendors must be careful about the introduction of applications that bypass the Android security model by using native code or irresponsibly high privilege levels. These applications should undergo the same level of code review and scrutiny currently used by engineers elsewhere in the software industry.

References

- http://code.google.com/p/android-scripting/
- http://developer.android.com/sdk/ndk/index.html#overview

- http://developer.android.com/reference/android/Manifest.permission.html
- http://en.wikipedia.org/wiki/Capability-based_security
- http://en.wikipedia.org/wiki/Java_Native_Interface

About IOActive