Hidden Android Permissions: Remote Code Execution and Shell Access using a Live Wallpaper

Qusay H. Mahmoud and Dylan Kauling
Dept. of Electrical, Computer and Software Engineering
University of Ontario Institute of Technology
Oshawa, ON, Canada

Shaun Zanin
School of Computer Science
University of Guelph
Guelph, ON, Canada

***Abstract*—Given Android's popularity, it is likely that Android devices will be targeted with increasing frequency by malware designers, in particular by programs which are designed to steal some of the sensitive personal or financial information. This threat puts an obligation on security professionals and software developers to work to ensure that the Android platform is as secure as possible. Users of mobile devices should also be aware that the trust they grant to mobile applications they download from app stores can be exploited. In this demo paper, we present a live wallpaper app to demonstrate how trust can be exploited to gain shell access even on unrooted Android devices.**

Keywords—Android; mobile apps; wallpaper; exploiting trust; shell access.

I. INTRODUCTION

Google created Android as a fork of the well-known Linux kernel, starting with version 2.6. While it did split from some of the later versions of the Linux Kernel (due to disputes over code maintenance), presently most of the kernel's codebase remains the same. Fortunately, the newest versions of Android are beginning to become merged with the upstream code base of the Linux Kernel.

Since Android shares most of the same codebase as Linux, it benefits from the large development community which has spent countless man-hours developing and testing Linux for the past 20 years. While it has shared a few of the security defects found in the Linux kernel, updates released by Google usually integrate the same fixes which have been added to the Linux source tree in order to make users' systems more secure. By using the Linux kernel as a codebase, Google helps to ensure that Android devices remain as secure as possible.

Android begins to diverge sharply from the normal Linux model in regards to its security model. It implements a system where every application installed on an Android device is "sandboxed" and prevented from accessing other applications/resources. This is explained in greater detail in [1]. Most recent privilege escalation attacks on Android have focused on inter-application privileges, rather than attempting to gain root access. Such an attack is demonstrated in [2].

Without root access, the attack is restricted to data stored outside of application directories, but a great deal of information can still be accessed from shared storage and exposed interfaces. The general idea is to use another application which is already installed on a user's device to manipulate resources by using permissions which it has already been granted legitimately [3]. The attacker's original application uses an exposed interface in another installed application in order to take certain actions which it has not been granted permission to execute itself. An example would be a wallpaper manager, which takes advantage of an interface left exposed by a developer on a power saving application. A power saving application has a plausible reason to need access to Wi-Fi permissions. If the power saving application uses the interface of another application (say connectivity manager) to control Wi-Fi connectivity, then the malicious wallpaper manager could use the exposed portion of the power manager in order to access and disable a user's Wi-Fi connectivity.

Android applications typically run inside the Dalvik Virtual Machine. The Dalvik VM is a virtual machine which is similar in conceptualization to the Java HotSpot system developed by Oracle. Significant structural differences between the two systems exist; however, many of the same Java APIs are available in both environments. An example of this is the Java Runtime object, which this paper uses as part of a newly discovered form of attack which allows malicious parties to mask their attack behind a legitimate application, running it as a native Linux process. This gives attackers shell access to a user's device, in addition to any other compiled executable which they deploy and run.

II. THE EXPLOIT

The exploit utilizes the Runtime component of the Java API. This object allows a Java application to run an ELF (Executable and Linkable Format) executable from within the Dalvik virtual machine. This is not part of the Java Native Interface (JNI) which is provided through the Android Native Development Kit. All the normal management features for applications which exist in the Dalvik VM, including the running process manager, do not interact with the native executable, which means that a user normally will not even be aware that the process will still be running. It is very difficult to terminate these processes without some kind of shell interface.

The attack outlined in this paper consists of a social engineering component, as well as a technical component. The social engineering component is the implementation of an initial "enticement" application that convinces a user to download it and install it. Enticing a user with certain features to then execute an attack is explored further in [4].

978-1-5090-6196-9/17/$31.00 ©2017 IEEE

If an application asks for excessive permissions at install time, some users will get suspicious and become hesitant to install it. In this case, a live wallpaper (based on one of the samples provided in the Android SDK) was chosen as an attack vector. In Android, this type of wallpaper could be easily modified to deliver the desired functionality. Many people would not think that a wallpaper is an "application", and users will be much more likely to install them than a regular application. Live wallpapers run in the background of a user's "home screen" when they are using their device. They can be given all the same security permissions as any other application running on the Android system; an example of the reason they are permitted such functionality can be seen in the implementations of some of the already existing wallpapers present on the Google Play store. For example, some wallpapers require both location information and Internet access in order to localize the type of weather being displayed. These same permissions, however, can also be used to discreetly track a user's location and send it to a remote server.

A. *Exploit Application Lifecycle*

The attack presented here utilizes the ability to run native executables in order to allow an attacker to gain shell access to the user's Android device. It uses netcat, a traditional UNIX application for connecting two machines via sockets. Netcat is cross-compiled from the Attacker's desktop system in order to run on Android, and is packaged as a regular resource within the APK file with which the application is installed. The Java-based wallpaper which the user sees launches Netcat in the background and connects it to a remote server, which then listens for connections from a victim's device. Upon connection, Netcat launches an instance of the Android system shell and pipes remote input from the TCP connection to the shell. This occurs without being displayed to the user at all. Once shell access is granted, the attacker can utilize any of the functionality which the application was originally granted; this gives attackers a huge degree of flexibility, in that their attack is: not visible to the user, and can be customized to do nearly anything, rather than just a specific task built-in to their application. Since it is running as a native Linux process, simply changing wallpapers does not close the tunnel. The Netcat-enabled shell runs until the application is uninstalled or the user reboots the device, as is shown in Fig. 1.

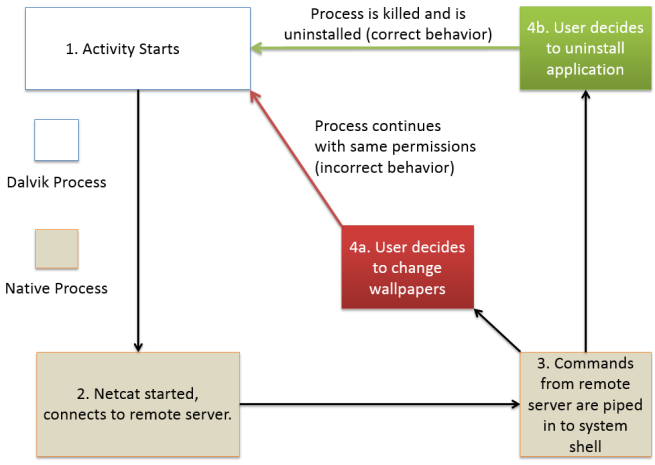

Fig. 1. Wallpaper exploit application lifecycle.

An added concern with this exploit is its increased efficacy from the social side on Android Marshmallow devices. As can be seen in Fig. 2 (right), the install permissions are all listed properly on the device running Lollipop, but mysteriously the Full Network Access permission is hidden from the installation dialog in Marshmallow, yet it is provided all the same. This can be verified by testing the exploit, as well as going into the application information, permissions, then hidden in the sub-menu for "All Permissions".

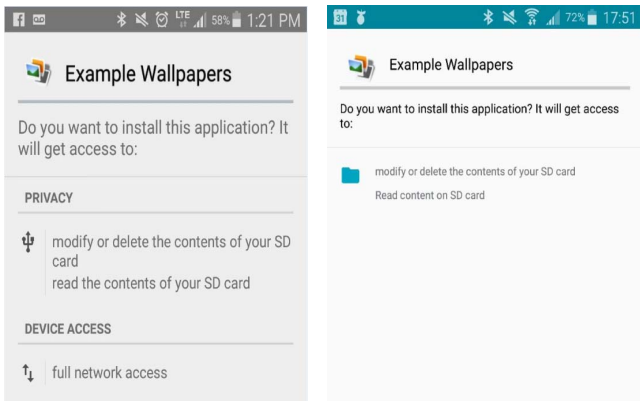

Fig. 2. Lollipop (left) vs. Marshmallow (right) installation.

III. Conclusion

The wallpaper app presented can give an attacker shell access to a user's device while not appearing on process-management tools provided by default on Android devices. The system shell allows attackers to run privilege escalation attacks using previously existing flaws in the Android security model in order to obtain more permission, without the user being aware of its operation. A re-examination of the permission model of the Android platform is required, with the full list of permissions being provided to the application for Marshmallow and above being the minimum goal. Also, any process started by an application should be terminated when it is no longer in use, unless they are registered as an actual service on the device and the necessary permissions are held. Unless changes are made to how the Android permission system works to allow for more fine-grained control, it is necessary for software developers to take careful attention and protect their application against exploitation and to ensure that the software they develop is as secure as possible. It is also the user's responsibility to be wary of any software they install on their device, whether from a trusted source or not, as granted permissions can give much more control over the device than one might think. Wallpaper demo and source code are available online at [5].

References

[1] S. Höbarth and R. Mayrhofer, "A Framework for On-Device Privilege Escalation Exploit Execution on Android," in Proceedings of IWSSI/SPMU, 2011.

[2] L. Davi, A. Dmitrienko, A. Sadeghi and M. Winandy, "Privilege Escalation Attacks on Android," in Proceedings of International Conference on Information Security, pp. 346-360, 2010.

[3] R. Mathew, "Study of Privilege Escalation Attack on Android and its Countermeasures," International Journal of Engineering Science and Technology, vol. 4, no. 9, pp. 4078-4082, 2012.

[4] T. Vidas, D. Votipka and N. Christin, "All Your Droid Are Belong To Us: A Survey of Current Android Attacks," In Proc. of the 5th Usenix Workshop on Offensive Technologies, pp. 81-90, 2011.

[5] Wallpaper demo and code, 2016. [Online]. wallpaper.nextproject.ca.