The User is the Vulnerability

Breaking the Android Security Model Through the Permissions System

Author:
Nathaniel Tenczar
ntenczar@gmail.com

Mentor:
Ming Chow

December 13, 2013

Abstract

The prevalence of the Android operating system today has led to Android devices ennding up in the hands of millions of users. Due to this, social engineering and exploiting the user has been an effective method of installing malware on mobile devices. This paper addresses a few methods of breaking the Android permissions system by taking advantage of the user's lack of attention to the permissions they enable when installing applications.

1 Introduction

Beginning in 2008 with the release of Android 1.0, the mobile operating system has found its way onto millions of devices and has been one of the few major competitors for Apple's mobile operating system: iOS [7]. Today, Android devices command more than half of the mobile device market at 52.2%, while iOS devices comprise 40.6% of the market [3]. The rise of Android to a large share of the market has lead to a greater concern for security, as users more heavily rely on their devices to transmit and store sensitive information such as private company data, banking information, and text messaging and email. The Android security model does its due diligence to protect the user from many vulnerabilities through its use of sandboxing, filesystem encryption, and application permissions, but as this paper will discuss, a large vulnerability is in fact created by the user itself, and is enabled by Android's application permissions system.

With a massive user base numbering in the hundreds of millions, the popular Android OS has reached a critical stage in any relatively young product's life cycle. By enabling software developers to write applications using the Android API, the floodgates were opened for generations of new native applications to be developed by a large number of different developers. In conjunction with this, Android allowed multiple distribution channels to be set up to deploy apps to mobile devices, the most prevalent of these channels being the Google Play store, maintained by Android/Google, and the Amazon App Store. The decision to allow multiple distribution channels constrasts that of iOS, where apps may only by purchased through the Apple maintained iTunes app store [1]. By allowing multiple distribution channels, Android opens itself to more potential vulnerabilities than iOS, but combats this by treating each application that is installed as a potentially malicious application. The OS does this by requiring an application to request access to permissions at install time, which a user must verify.

The Android permissions system is derived from the Linux permission system, although in Android, each application is given its own Linux user ID and group ID, and aspects of the system have their own distinct identities. [2]. In Android, permissions work by assigning permissions to system resources and API calls which the user must allow access to when an application is installed. There are three permission levels that correspond to threat levels that resources/API calls fall into: normal permissions, dangerous permissions, and signature/system permissions [4]. The Android API ensures that an application requesting a certain permission actually has access to that permission by invoking a permission validation process that is a part of a system process.

2 To the Community

Due to the fact that the market is so heavily saturated with devices running the Android OS, the common man actually using the devices must have some notion of how to protect himself against potential security issues. As outlined earlier, Android does its part to protect the user against security threats posed by benign programs, or vulnerabilities in applications created by trusted developers. Instead, however, the common user must be vigilant and somewhat judicious when installing applications on their device, especially if an application comes from a third party developer from a less-than-reputable distribution channel. In an article detailing a particular brand of Android malware dubbed "Perkele", which will be further discussed in Section 3.2, Brian Krebs states that "a modicum of common sense and impulse control

can keep most Android users out of trouble." [6] This paper addresses the idea that Android users must be aware that when installing an application, they must assess whether or not the permissions they are about to enable for a that application are actually necessary for that application's functionality.

The idea that permissions should be examined judiciously also applies to equally to application developers. When designing a new application, the onus is on the developer to choose to enable only the permissions that the application actually needs, following the principle of least privilege. In various stages of an application's development cycle, certain features and permissions may be modified or added/removed, and thus overprivileging is a possible side effect. An overprivileged application increases its susceptibility to being exploited, and thus the developer too should be judicious when deciding an application's allowed permissions.

3 Action Items

3.1 Exploiting the Inattentive User

Even in very simple and seemingly obvious cases, it can be easy to fool the user into giving an application permissions that it simply does not require. Here, I have created an application as a proof-of-concept that solely intercepts and displays the user's text messages, and completely complies with the Android permissions system. The source may be viewed on Github [9].

Dubbed, "Bogus App" this application doesn't do all that much, but shows that any app may be extended to feature this same functionality: interecepting text messages. The heart of this app relies on two things: the permissions request in the Android Manifest (`AndroidManifest.xml`), and the `SmsReceiver`, an instance of the `BroadcastReceiver` class shown in `SmsReceiver.java`.

The Android Manifest contains the all-important lines describing what permissions that the application will request when it is installed. This can be seen in the `<uses-permission>` tags, and in the case of Bogus App, the permissions for `android.permission.RECEIVE_SMS` are requested.

Bogus App continues to define an `SmsReceiver` that extends a `BroadcastReceiver` which waits to receive text messages from an `Intent` (more on those in Section 3.3) The messages are then retreived from the `Bundle` (an Android storage mechanism) inside the `Intent`, and sent to the `updateMessageText` method of `MainActivity`. This method in `MainActivity` merely appends the text message and its metadata to a text view, which is the only view accessible from the app.

Even for an Android development novice, there is very little to do to build a simple app with this functionality. A more seasoned developer might note that all of this could be very easily baked into another application, perhaps even to mask its true purpose. A particularly sinister use of this could utilize the text message interception functionality, sending intercepted text messages to a remote server, and finally hiding all of this behind a seemingly legitimate application. This could then be distributed through a 3rd party distribution channel to make it even easier to prey on users.

This relates back to the idea that an inept user may install any application, regardless of warnings from the Android system that the application is requesting additional functionality that seems out of place. For instance, if the malicious app described in the previous paragraph is actually a "flashlight" app, and is requesting permissions to receive text messages and access the internet, this may seem wholly unncessary. However, if the user is not paying attention to the permissions as they install an app, this can go completely unnoticed, and the malicious app will easily slip through the built in Android security measures that would normally prevent it.

3.2 Perkele: Fooling the Alert User

There are, however, ways that even a user paying close attention to application permissions may be fooled.

Perkele, a malware application designed to target people accessing their banks online, hits the Android security model at its weakest point: the user. [6] The attack is twofold: first, the victim first visits the

website of their bank via their mobile web browser. At this point, the website has already been exploited, and the victim is subjected to the effects of a script that prompts the user to install an application that is described as an additional security mechanism from the bank. At this point, the user has been tricked into thinking that the bank does require additional security, and will likely install the application.

There is more to the Perkele exploit which requires a combined effort of cross-site scripting, the application itself, and malicious web servers, but the main point here is that users will probably install almost any application if they deem it necessary without too much premeditation or thought as to whether the application has been created by a trusted developer, or if it is actually an application that the user needs.

This exploit will likely even work on the most alert of users. Due to the fact that the request appears to be coming from a seemingly legitimate source, a bank, even a well informed user that only allows certain applications certain permissions may fall into the trap of giving malware the necessary permissions to perform its exploit.

3.3 The Android Permissions System May Already Be Broken

The ideas of both an inept and astute user have already been covered, but what if the permission system is already broken? The permissions system for the Android model stipulates that an application only has access to the permissions that the application requests at install time, but what if there is another way for an application to access other privileges, in a malicious or even benign manner?

The work of Orthacker et al. [8] illustrates a few additional ways in which the permissions model can be broken to allow additional permissions to be utilized by an application which was not previously allowed those permissions. Orthacker et al. details a way in which applications can effectively collude to provide permissions to one another using built in Android APIs that are unaware of the permissions system. For example, one application may request GPS permissions and seem legitimate, as it is an app that normally utilizes GPS coordinates, such as one that provides maps. This GPS app can provide its GPS services to any other application that it wants to, even if the application requesting the GPS services does not have GPS permissions. This can be extended for virtually any permission that the Android API provides.

This permissions sharing exploit utilizes Android API functionality, namely, inter-process communications. These communications include the `Intent`, which describes operations to be performed by the receiving party, which include Activities, the basic building blocks of applications that describe how an application displays its views and performs actions with services or API calls. Intents are passed to Activities, and from the Intent the Activity can use the `getExtras` method to retrieve from the Intent what functionality that the previous Activity or process is requesting.

While applications can be restricted from accessing certain permissions, there is a very easy way around this in the form of having applications collude in a way that an under-permissioned app receives the permissions it requires from anotehr app which happens to have the permissions.

4 Conclusion

In this paper, multiple ways to circumvent or abide by the Android permissions system were discussed. While circumventing the permissions system by enabling applications with different permissions to collude is certainly a possibility for those wishing to exploit the system, an easier method is to go right after the user. Users of the Android permissions system will often blindly ignore permissions requests and grant any permissions and app requests. Felt et al. [5], which also examined the effectiveness of Android permissions as a part of the Android security model, asserted that "installation security warnings may not be an effective malware prevention tool, even for alert users." There is certainly psychology research that needs to be done to evaluate how to get users to respond to the security warnings presented by permissions requests, with the end resulting being one in which the Android security model is re-tooled

to better accomodate and inform users when they are deciding to grant an app permissions.

References

[1] Mobile security - android vs. ios. http://www.veracode.com/resources/android-ios-security, December 2013.

[2] Permissions - android developers. https://developer.android.com/training/articles/security-tips.html, December 2013.

[3] Michael Calia. Apple's market share grows. http://online.wsj.com/news/articles/SB10001424052702303497804579240380791869444, December 2013.

[4] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[5] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

[6] Brian Krebs. A closer look: Perkele android malware kit. http://krebsonsecurity.com/2013/08/a-closer-look-perkele-android-malware-kit/, December 2013.

[7] Dan Morrill. Announcing the android 1.0 sdk, release 1. http://android-developers.blogspot.com/2008/09/announcing-android-10-sdk-release-1.html, September 2008.

[8] Clemens Orthacker, Peter Teufl, Stefan Kraxberger, Günther Lackner, Michael Gissing, Alexander Marsalek, Johannes Leibetseder, and Oliver Prevenhueber. Android security permissions - can we trust them? In Ramjee Prasad, Károly Farkas, Andreas U. Schmidt, Antonio Lioy, Giovanni Russello, and Flaminia L. Luccio, editors, *MobiSec*, volume 94 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 40–51. Springer, 2011.

[9] Nate Tenczar. Bogus app. https://github.com/ntenczar/BogusApp, 2013.