Android Applications, Can You Trust Google Play on These

Ge Gao
Comp 116 Security Final Paper
Professor Ming Chow
December 5, 2016

Table of Contents

Abstract

Mobile devices have become such a natural extension of people's day-to-day life. Based on the numbers provided by Statista, by the end of September 2017, there are over 3,300,000 Android applications hosted on Google Play Store[1]. Downloading an Android application onto the local device takes nothing more than 3 or 4 finger actions, a fact that is both convenient and potentially hazardous. Just because an application is hosted on Google Play Store is not a green pass to its security. To most users of an Android applications, they have absolutely no knowledge of how much of their activities spending on the application is being monitored or where are all of their personal data are flying to. They are not aware of what measures Google has taken, if any, in reviewing and monitoring the application's safety before it comfortably lies in the users' devices. This paper primarily focuses on the current shortcomings in Google Play's application monitoring process and potentially malicious applications can exploit such vulnerabilities and bypass the security measures on Google Play. What's more, as a user, how to avoid being harmed by malwares that Google failed to detect.

Introduction

In order to get an insight on why we are suffering what we are suffering with the universality of Android malwares and data stealing, it is imperative to understand where are the vulnerabilities in our first line of defense, the Google Play Store, and the various ways malicious applications utilize and gain values from them. This paper provides detailed case analysis of two real Android malwares, each serves as a representative of the attacking patterns of its kind, that have made noticeable damages. It provides an insight into the how the current policy of application uploading and defense mechanisms on Google Play Store are fundamentally

vulnerable through the lens of malwares that easily made their ways into users' actual devices through Google.

To the Community

By the second quarter of 2016, Android is taking up 86.2% of the market share of smartphone operating system[2]. With Android's notorious reputation in security and protection of data privacy, the severity of the problem is significantly attributed to the overly large amount of trust placed in Google Play Store, holding ourselves responsible to the current state. Sadly, for most people, it is easier to live under the seemingly soothing idea of "Only Downloading Apps from Google Play Store and Nowhere else" and conveniently believe that such "security measure" will keep them away from the malicious world of malware and data stealing than to actually learn what is happening behind the scenes when an application is uploaded to Google Play Store and check the various types of permissions regarding sensitive data the application has. Here is cold and pathetic truth: more often than not, people should be holding their selves responsible when the bad things happen.

1. Just How Bad is the Issue

By September 2017, according to researchers from Check Point, Google Play has been hit by another flood of Android malwares with as many as 21.1 million infections from one malware family. In the same month, a new strain of Android malware called "Expensive Wall" was found in more than 50 applications on Google Play. The cumulative downloads of this malware were approximately 4.2 million times[3].

2. Thinking in Enemy's Hat

a. Harm You Later in Real World

In the spirit of keeping all potentially hazardous testing activities separate from the real devices, one strategy people might try is to run a target Android emulator in a virtual environment that creates the false feeling of having validated the safety of the application. Attackers are clearly aware of how to exploit such psychology among people. The following java snippet from *Figure 1* was generated from the APK of a malicious banker Trojan application that infected up to 250 devices[4].

```
if ((Build.MODEL.contains("google_sdk")) ||
    (Build.MODEL.contains("Emulator")) ||
    (Build.MODEL.contains("Android SDK")) ||
    (Build.FINGERPRINT.startsWith("generic")) ||
    (Build.FINGERPRINT.startsWith("unknown")) ||
    (Build.MODEL.contains("Android SDK built for x86")) ||
    (Build.MANUFACTURER.contains("Genymotion")))
{
  i = 1;
  if ((!Build.BRAND.startsWith("generic")) || (!Build.DEVICE.startsWith("generic"))) {
    break label155;
  }
}
```

Figure 1

When starting the application, the malware simply checks if the application is being run in an emulator setting. If so, it will hide all the following malicious activities to give testers a fake sense of confidence and assurance. As the testers let down their guards and installed the application to their actual devices, as *Figure 2* shows, the application secretly sends out all the device information to a remote server and continue to obtain administrator access of the phone[5].

```
localObject = "" + "urlb:1, ";
localObject = (String)localObject + "imei:" + this.jdField_a_of_type_AndroidTelephonyTelephonyManager.getDeviceId() + ", ";
localObject = (String)localObject + "country:" + this.jdField_a_of_type_AndroidTelephonyTelephonyManager.getNetworkCountryIso() + ", ";
localObject = (String)localObject + "cell:" + this.jdField_a_of_type_AndroidTelephonyTelephonyManager.getSimOperatorName() + ", ";
localObject = (String)localObject + "android:" + Build.VERSION.RELEASE + ", ";
localObject = (String)localObject + "model:" + Build.MODEL + ", ";
localObject = (String)localObject + "phonenumber:" + this.jdField_a_of_type_AndroidTelephonyTelephonyManager.getLine1Number() + ", ";
localObject = (String)localObject + "sim:" + this.jdField_a_of_type_AndroidTelephonyTelephonyManager.getSimSerialNumber() + ", ";
localObject = (String)localObject + "app:null, ";
localObject = (String)localObject + "hsmsm: " + s.a(this.jdField_a_of_type_AndroidContentContext) + ", ";
localObject = (String)localObject + "hadmin: " + n.a(this.jdField_a_of_type_AndroidContentContext) + ", ";
localJSONObject.put("info", (String)localObject + "ver:6.0.8");
```

Figure 2

The server can then respond with all different kinds of malicious instructions for the application to further perform. Separating an attack into different stages using a remote C&C (command and control) server and wait for further instructions like such helps the malware blends into the sea of applications on Google as it appears extremely clean when first published to Google Play, and all the malicious actions then follow after the malware has made its way to user devices in its short yet impactful survival on Google Play.

b. Bounce the Bouncer with Persistence

It is not fair to say that Google has done no significant counter measures against the prevalence of malwares on Google Play. One of the effective efforts was the launch of the security service for Android malwares called Google Bouncer in February 2012. It silently scans all Android applications and potentially dangerous developer accounts on Google Play with its reputation engine. Based on the statistics by Google, Bouncer is responsible for a 40% drop of malwares across the entire Play Store[6]. However, attackers are still able to bounce back the security checks from Google Bouncers with unbearably simple techniques, a fact that is disturbing to the community.

In 2015, Check Point Mobile Threat Prevention detected a malware called Brain Test that had up to 500,000 downloads each in two of its instances[7]. Brain Test was able to bypass the security checks from Google Bouncer by performing an easy check of the origin of where the application is being run. From the smali assembly code block in Figure 3[4], we notice that the malware first converts the source URL into all lower-case string to take care of all the permutations of capitalization and then simply checks if the source contains keywords similar to "google", "system", and "android" that potentially signify that the application is being run from a google IP address.

```
.method private isFilter(Ljava/lang/String;)Z
    .locals 2
    invoke-static {}, LA001;->a()Z
    move-result v0
    invoke-static {v0}, LA001;->a0(I)I
    nop
    const/4 v1, -0x1
    invoke-virtual {p1}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;
    move-result-object p1
    const-string v0, "android"
    invoke-virtual {p1, v0}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I
    move-result v0
    if-ne v0, v1, :cond_0
    const-string v0, "system"
    invoke-virtual {p1, v0}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I
    move-result v0
    if-ne v0, v1, :cond_0
    const-string v0, "google"
    invoke-virtual {p1, v0}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I
    move-result v0
    if-ne v0, v1, :cond_0
    const-string v0, "samsung"
    invoke-virtual {p1, v0}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I
    move-result v0
    if-eq v0, v1, :cond_1

    :cond_0
    const/4 v0, 0x1

    :goto_0
    return v0

    :cond_1
    const/4 v0, 0x0

    goto :goto_0
.end method
```

Figure 3

Superficially camouflaged as a harmless application that tests users' IQ, Brain Test is gaining root access of the devices and dynamically downloading files and executable code that

facilitate any criminal actions imaginable. Worst of all, when the damage happens, it is almost impossible for the victim to get rid of the malware. We can see from the Java code snippet generated from JD-GUI in *Figure 4* that the application is watching for the event PACKAGE_REMOVED, if the user tries to remove the application, with root access to the device, the application simply reinstalls the itself.

```
if (str.equals(MainApplication.C))
{
  f.J(paramContext);
  return;
}
if ((str.equals("android.intent.action.PACKAGE_REMOVED")) && (f.C(paramContext, paramIntent))) {}
```

Figure 4

In the end, it took Google five days to remove the malicious Brain Test from Google Play after Check Point first reported to Google on September 10, 2015. Sadly, 500,000 downloads had happened during this period.

3. Making it Better

a. How Does Google Improve the Situation?

i. Human Monitoring Is Necessary

The appearance of algorithmically complex and automated machine-monitoring tools like the Google Bouncer has inarguably alleviate the infamous security issue on Play Store. However, the fact that Bouncer can be easily bypassed by the unbearably simple technique in the malware mentioned above and the fact that it can take as short as a few minutes to upload an application to Google Play

and accessible and downloadable by billions of users is disturbing to anyone. While Google loves to automate the review process of an application, it is necessary to have humans monitor applications uploaded to Google Play to ensure no malicious activities are happening behind the scenes and the application's description is in accordance with what the code actually does. A day will certainly come when the sophisticated algorithm is comprehensive enough to cover everything that a human can identify in an application. However, until that day, human monitoring is indispensable.

b. We Must Protect Ourselves

i. Fragmentation Haunts Us, Do Not Fall Behind

Google is well aware of the severity of the prevalence of security concerns raised by Android malwares on Play Store. As notorious as the security on Google Play is, Google's efforts towards alleviating the problem can be noticed. Starting from August 2015, Google announced that it would release over-the-air security patches every month in additional to the routine OS updates[8]. However, these efforts were not effectively reflected in the users' devices. Unlike Apple where almost all of the devices are made by Apple, Google's operating system is being run on a variety of devices by different carriers. Fragmentation brewed under such model makes it hard for Google's patches and updates to get to the users in time. So as hard as it can be, users should try to constantly check for and sometimes manually obtain the newest Android updates and protect themselves with the latest defenses and vulnerability fixes.

ii. Think Again Before You Ask for Power

Customers rooted their Android devices to have an even more "open" environment where they are given more power as administrators to achieve functionalities like getting rid of bloatware, control the CPU task management, and remove advertisement. However, great power comes with extremely high risk. The first important action that most of the Android malware discussed above is to obtain the root access of the devices in order to perform more impactful damages and criminal activities, as shown by above malware examples. Rooting the Android devices saves malwares the trouble of obtaining the administrators' access, constantly presenting the devices vulnerable to malicious applications. If you do not have a good reason to root your device, do not expose yourself to harms.

Conclusion

Customers are drawn towards the open-source OS that Android provides, a fact that can be seen from that Android is taking up 86.2% of the entire worldwide market share in the OS market. As Android is gradually taking up the majority of the market, the security issues on Google Play have now become more important than ever. While users are empowered with the open development environment that Google provides for Android, it is important to realize that such trust placed in both costumers and hardware manufacturers and carriers unavoidably creates opportunities for malwares and attackers. As seen the cases analysis of this paper, it is never enough to just follow "download apps only from Google Play" and sit back and relax, a current truth that is both disturbing and sad. While Google is making progress to improve the current situation such as launching automated antivirus system like Google Bouncer and including more human supervision in code review, we cannot let our guards down and must actively take steps,

including keeping the security patches and OS version updated and avoid obtaining root access of the device to protect ourselves as costumers. It is important to Google to find a balance between freedom and security. We can keep an optimistic attitude that a day will eventually come when neither is compromised.

Bibliography

[1] AppBrain. *Number of available applications in the Google Play Store from December 2009 to September 2017*. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ (accessed December 12, 2017).

[2] *Ltd, Moon Technolabs Pvt. "Apple Vs Android-A Comparative Study 2017."* AndroidPub*, AndroidPub, 1 Mar. 2017, android.jlelse.eu/apple-vs-android-a-comparative-study-2017-c5799a0a1683.*

[3] "How Malware Keeps Sneaking Past Google Play's Defenses." Yahoo! Finance, Yahoo!, 22 Sept. 2017, finance.yahoo.com/news/malware-keeps-sneaking-past-google-120000474.html.

[4] *Ashishb. "Ashishb/Android-Malware."* GitHub*, 6 Mar. 2017, github.com/ashishb/android-malware.*

[5] Piskáček, Jan. "Android Banker Trojan Preys on Credit Card Information." *Avast Blog*, 5 May 2016, blog.avast.com/android-banker-trojan-preys-on-credit-card-information.

[6] Hou, Olivia. "A Look at Google Bouncer." TrendLabs Security Intelligence Blog, 20 July 2012, blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/.

[7] Polkovnichenko, Andrey, and Alon Boxiner. "BrainTest - a New Level of Sophistication in Mobile Malware." Check Point, 21 Sept. 2015, blog.checkpoint.com/2015/09/21/braintest-a-new-level-of-sophistication-in-mobile-malware/.

[8] Lardinois, Frederic. "Google And Samsung Will Now Release Monthly OTA Android Security Updates." TechCrunch, TechCrunch, 5 Aug. 2015, techcrunch.com/2015/08/05/google-and-samsung-will-now-release-monthly-ota-android-security-updates/.

[9] T., Nick. "10 Cool Things You Can Do with a Rooted Android Smartphone." Phone Arena, PhoneArena, 10 Oct. 2012, phonearena.com/news/10-cool-things-you-can-do-with-a-rooted-Android-smartphone_id35360.