Android Plugin Becomes a Catastrophe to Android Ecosystem

Cong Zheng, Tongbo Luo, Zhi Xu, Wenju Hu, Xin Ouyang
Palo Alto Networks, CA, USA
{cozheng, tluo, zxu, whu, xouyang}@paloaltonetworks.com

ABSTRACT

The Android Plugin is a new application-level virtualization technology in Android system. Android Plugin allows a host app to create a virtual environment, in which any other APK files can be directly launched as runnable plugins without the installation. Unlike the dynamic code loading, the plugin-enabled host app provides a proxy layer between plugin apps and the Android framework. This virtualization technology has been applied in the development of hot apps, such as the "Parallel Space" app. However, the Android Plugin technology has completely changed the landscape of Android ecosystem security. We will demonstrate our perspectives by proposing some attacks via Android Plugin: a) A zero-permission app can bypass the permission check and the data isolation mechanism by exploiting two vulnerabilities we discovered in Android plugin frameworks. b) A new Android phishing attack allows attackers to phish any target apps at no cost. c) The current app promotion system can also be compromised by attackers through directly launching as many as promoted apps in the plugin environment. d) With our developed tool "Z4Plugin", attackers can easily transform any malicious APK file to a new APK file, which can evade all engines in VirusTotal. At last, we have proposed mitigation solutions for above attacks.

1 INTRODUCTION

Virtualization can be viewed as part of an overall trend in every aspect of contemporary technologies, which has been applied to a wide range of system layers from the hardware-level to the OS-level. In the nutshell, every virtualization technology basically provides an additional layer of abstraction and automation of the true underlying hardware or software to extend beyond the underlying architecture. In spirit of this fact, it is not surprising that mobile platforms also take full advantage of this trend. In the past years, we have observed an application-level virtualization technology called "Android Plugin", utilized in the Android ecosystem. Android Plugin has become more and more popular in the mobile application development community.

The Android plugin technology is an innovative application-level virtualization for Android apps. It was originally developed for purposes of hot patching, reducing the released APK size and others. With the enforcement of Android Plugin technology, an app (called "host" app) can directly load and launch any apps (called "plugin" apps) from their APK files without the installation [12]. This black magic brings a groundbreaking way to launch an independent app, which actually has not been installed in the system. It is of no doubt that the plugin technology departs from the original Android design that every launched app must have been installed in the system. The common application scenario of the Android Plugin technology is launching multiple instances of same app on the same device. For example, users can login two Twitter accounts (e.g. one for personal and one for business) simultaneously. In Google Play, the most popular plugin app is "Parallel Space" [5], which has been installed by 50-100 million users. The popular social app "Weibo" [8] also uses the plugin technology. In Google play, there are 51 apps using the open-source plugin frameworks. So, the Android Plugin technology has been deployed in some famous apps and indirectly used by millions of users.

However, the security risks of Android Plugin have been underestimated. In this paper, we demonstrate our perspectives by proposing some attacks via Android Plugin, as follows:

- *Attack by a zero-permission app*: once a malicious zero-permission app is loaded as a plugin, it can use all Android permissions and steal private data of other plugin apps.
- *New Phishing Attack*: the attacker can launch the authentic apps (e.g. banking apps) without any modifications in the Android Plugin environment, and hijack user inputs.
- *App Promotion Fraud*: with the plugin technology, the host app can launch arbitrary promoted apps without the user consensus for easily making profits.
- *Evasion by Z4Plugin*: we proposed the Z4plugin tool to transform any Android malware to its evasive version, which bypasses all AV engines in Virustotal in our test.

2 BACKGROUND

2.1 Technical Basics of Android Plugin

Before we discuss the technical parts of Android Plugin technology, it should be noted that the Android Plugin is totally different with the Android dynamic code loading. In the dynamic code loading, an app can load .jar, .dex or .so files at runtime. Interfaces between the main code and the dynamic code should be well predefined. However, in terms of the Android Plugin technology, an app can directly load and launch arbitrary .apk files. Since the plugin virtual environment is very universal, no interfaces are needed anymore.

In the implementation of a Android Plugin runtime environment, there are five basic concepts and mechanisms:

- *Shared UID.* All plugin apps share the same UID with the host app.
- *Pre-defined stub components and permissions.* The host app has pre-defined components and permissions for plugin apps.

- *Dynamic Proxy Hook.* The host app has hooked API invocations of plugin apps by the dynamic proxy technique, so that the Android system thinks that all API requests and components are from the host app.
- *Resource loading.* As the plugin app is not installed in the system, the host app must take over the procedure of loading app resources in the plugin app process.
- *Component Lifecycle Management.* When the component in the plugin process is ready to be destroyed, the corresponding stub component should also be destroyed simultaneously.

2.2 Open Source Android Plugin Frameworks

The two most popular open-source Android Plugin frameworks are "DroidPlugin" [1] and "VirtualApp" [2], both have adopted above five design principles to achieve the same goal. In addition to these two open source frameworks, some commercial apps developed their own, but similar plugin frameworks.

3 ATTACKS VIA ANDROID PLUGIN

As mentioned above, the most popular scenario is launching multiple instances of same app on the same device, especially for social apps. In this scenario, the host app is legitimate and it can launch any unknown apps as plugins. We have discovered two vulnerabilities in current plugin frameworks and security risks for users. By exploiting vulnerabilities, attackers can use a zero-permission app to bypass the permission model and the data isolation mechanism.

3.1 Attack by a Zero Permission App

We have analyzed DroidPlugin and VirtualApp frameworks, and conducted a comprehensive security evaluation on 18 popular host apps in Google Play. After that, we discovered two vulnerabilities within Android plugin frameworks. These two vulnerabilities can be exploited to bypass the Android permission model and the data isolation respectively.

Vulnerability 1: Lack of Android Permission Check. An app, once loaded as a plugin, can use all stub permissions without users' authorization. In this way, users face the new security threat from the zero-permission app, which can even become a powerful malware to invoke any sensitive APIs for different attacks. The vulnerability reason is that the host app shares all stub permissions with plugin apps, but it does not check permissions declared by plugin apps.

Vulnerability 2: Lack of Data Isolation. An app, once loaded as a plugin, it would steal any private data in other plugin apps. In the plugin context, the private data of a legitimate plugin app will never be secure, because the legitimate app does not run in an exclusive sandbox. The vulnerability reason is that the process of host app shares the same UID with processes of plugin apps. Further strict access control mechanism is missing in the current plugin frameworks.

It is concluded that the plugin host app, if carelessly implemented, could make users and users' data in a highly risky environment. This unsecured environment is no longer protected by the Android sandbox environment. Therefore, in the unrestricted Android plugin environment, a malicious plugin app with the zero permiossion can use any dangerous permissions and also easily steal the private data from other benign plugin apps, such as Facebook app and Gmail app.

3.2 New Android Phishing Attack

The phishing attack in the Android platform has never become such easier once the Android Plugin technology is abused. Previously, the phishing attack requires a fake app and a fake login Activity to lure users to install and input the credentials (e.g. Xbot Trojan [4]). However, some challenges are always existed in the phishing attacks. First, the attacker must develop an authentic or undistinguished login Activity to lure users. If the Activity layout is complicated, it is not easy to develop an exact same Activity. Second, the attackers should make the phishing attack inconspicuous, so that victims will not change the password immediately or later. To bypass above challenges, the static repackage [13, 16] is an alternative approach to implement the phishing the attack effectively. But, the static repackage still requires many manual reverse-engineering works in practice.

The Android Plugin technology makes the phishing attack very easy by nature. To achieve it, the attacker develops a general malicious host app, which has a module to hijack the input text information. Then, once the malicious host app launches any bank or social apps, it can directly steal the username and password. The advantages of this new phishing attack through abusing the plugin technology include: 1) It's scalable to launch the phishing attack to any apps. 2) Users cannot be aware of this attack as they actually play with the original app. 3) The development cost is low since no reverse-engineering is necessary.

3.3 Attack the App Promotion System

In this paper, we will demonstrate that the Android virtualization technology can be used to compromise the CPI-based app promotion system. We found that Android Plugin functionality becomes an innovative way to promote apps. A plugin-enabled app has the ability to automatically launch different apps in the virtualization environment without installing them. This provides a shortcut for attackers to make revenue from CPI-based ad networks as the promoted app can be launched on without any user interaction.

To promote apps, Adware developed by attackers commonly downloads apps and then frequently displays the app installation GUI to the user [9]. Clean Doctor [6], one of our captured samples, takes a different tactic to accomplish the same goal. Clean Doctor stealthily downloads many promoted apps from a cloud storage service. It does not request users to install these downloaded apps but instead launches the apps automatically. Indeed, the Android virtualization technology helps the Adware solve a huge problem in the past that the promoted apps are clicked and installed by users seldom.

This type of app promotion can also post security risks because of the comparatively weak security mechanisms used in current plugin frameworks. These plugin frameworks lack the ability to separate permissions and isolate data amongst different plugin instances. Thus, when a promoted app is executed through the plugin framework, it has the same permissions as the host app (typically all Android permissions) and can access the data of the host app or other plugin apps.

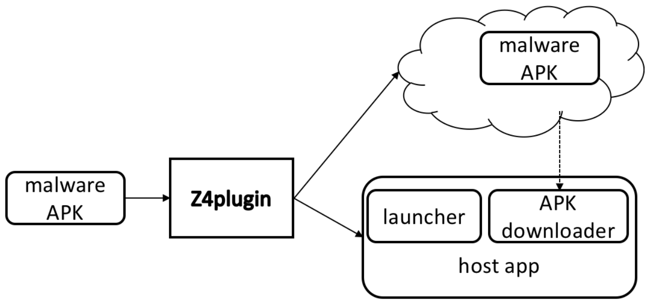

Figure 1: Z4plugin

3.4 Evasion Attack by Z4plugin

We have developed a tool, named "Z4plugin" (in Figure 1), which can effectively re-compile any Android malware to the new version, which can evade all engines in VirusTotal. The principle design of Z4plugin is dynamically launching malicious APK files in the virtualization environment hosted by a benign Android Plugin app. The benign host app uses an open-source Android Plugin framework, and includes one customized launcher to automatically launch the plugin app. Z4plugin also has a downloader module to fetch the original malware APK file from the cloud. Through our testing, the new generated Android malware (the host app) can easily bypass the detection of VirusTotal.

Like the dynamic code loading, the Android virtualization technology has the evasion ability towards the static analysis by nature. But, the virtualization technology can achieve more flexibility to design and update the "remote code". Because, the remote code is an arbitrary APK file in the Z4plugin design. Malware developers can easily design the structure of the remote APK file and do not need to pre-define any interfaces in the host app.

Because of the convenience provided by the plugin technology, Android malware could be simpler, smaller and remotely distributed in cloud. A bunch of Android malware samples as plugin apps can quickly be launched in victims' devices in schedule. Apparently, spreading malicious apps becomes much easier than other traditional approaches. If attackers start to use this new development model, there will be a huge challenge for the current static analysis engine commonly used in the Anti-AV industry.

The Android virtualization technology opens a new gate for Android malware. At the time of writing, we have captured and analyzed more than 182,000 Android malware, which abuse the open-source "DroidPlugin" or "VirtualApp" frameworks. Most of them take advantage of the DroidPlugin framework, which counts more than 180,000 in total. In November 2016, we discovered a new Android Trojan family named "PluginPhantom" [3], and utilizes a host app to schedule and control nine plugins. With the new architecture, PluginPhantom achieves more flexibility to update its modules without reinstalling apps. PluginPhantom also gains the ability to evade the static detection by hiding all malicious behaviors in plugins. The PluginPhantom actually was evolved from another Trojan family that didn't use the plugin technology. So, the PluginPhantom case shows that malware developers are using the Android Plugin technology as an alternative for the evasion.

4 MITIGATIONS

To mitigate above threats from the Android Plugin, we propose several practical solutions:

For patching vulnerable benign host apps: 1) *Check Permission*: To prevent the permission escalation of plugins, host app should only grant the permissions declared in plugin's manifest file. One of the easiest ways is to drop the permissions that are not declared by the plugin app when launching the plugin in a separate process. 2) *Process Capability Restriction*: Since the plugin app's process has the same UID, each plugin app has the capability to visit other plugins' private data. So, a potential solution is to restrict the capability for each plugin process. Seccomp [7, 11] is a capability system supported by the linux kernel, which has been deployed in the latest Android system. With the capability restriction, each plugin process can only access its own data directory.

For detecting malicious and grey apps, we can build a dynamic analysis sandbox [10, 14, 15] to detect malicious or suspicious behaviors. This dynamic analysis sandbox is capable of monitoring behaviors at different layers including Java and native frameworks. For example, behaviors including file operations, cryptos, privacy leakage, SMS sending, network traffic, etc. are monitored during an app's run-time. With these behaviors monitored and data captured, we have the ability to outline the app's behaviors and determine if the app is malicious or not. For the installed plugin apps, we have to monitor the activity of plugin processes, file operations in the plugin data directory, as well as the network traffic. Hence, we can combine the behavior information from both the host app and plugin apps, which help us build a better understanding of the analyzed app.

5 CONCLUSION

As the first one to deeply analyze security risks introduced by the Android Plugin technology, we discovered two security vulnerabilities in highly popular Android apps. We also have proposed a bunch of new attacks via Android Plugin: the evasion attack through our "Z4plugin" tool and the new phishing attack. We revealed the attack to compromise the app promotion system by abusing the Android Plugin. Therefore, before continuing to use the Android Plugin technology, these security problems must be addressed, and some of them can be mitigated by our proposed solutions.

REFERENCES

[1] 2015. DroidPlugin. https://github.com/DroidPluginTeam/DroidPlugin. (2015).
[2] 2015. VirtualApp. https://github.com/asLody/VirtualApp. (2015).
[3] 2016. PluginPhantom. http://researchcenter.paloaltonetworks.com/2016/11/unit42-pluginphantom-new-android-trojan-abuses-droidplugin-framework/. (2016).
[4] 2016. Xbot. http://researchcenter.paloaltonetworks.com/2016/02/new-android-trojan-xbot-phishes-credit-cards-and-bank-accounts-encrypts-devices-for-ransom/. (2016).
[5] 2017. Parallel Space. https://play.google.com/store/apps/details?id=com.lbe.parallel.intl&hl=en. (2017).
[6] 2017. Plugin Adware. http://researchcenter.paloaltonetworks.com/2017/03/unit42-new-trend-android-adware-abusing-android-plugin-frameworks/. (2017).
[7] 2017. Seccomp-BPF. (2017). http://lwn.net/Articles/475043/
[8] 2017. Weibo. https://play.google.com/store/apps/details?id=com.sina.weibo&hl=en. (2017).
[9] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable Third-Party Library Detection in Android and Its Security Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 356–367. https://doi.org/10.1145/2976749.2978333
[10] Antonio Bianchi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. 2015. NJAS: Sandboxing Unmodified Applications in Non-rooted Devices Running Stock Android. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '15)*. ACM, New York, NY, USA, 27–38. https://doi.org/10.1145/2808117.2808122
[11] Taesoo Kim and Nickolai Zeldovich. 2013. Practical and Effective Sandboxing for Non-root Users. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC'13)*. Berkeley, CA, USA, 139–144.
[12] Tongbo Luo, Cong Zheng, Zhi Xu, and Xin Ouyang. 2017. Anti-Plugin: Don't let your app play as an Android plugin. In *Proceedings of Blackhat Asia 2017 (Blackhat Asia'17)*.
[13] Chuangang Ren, Kai Chen, and Peng Liu. 2014. Droidmarking: Resilient Software Watermarking for Impeding Android Application Repackaging. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. ACM, New York, NY, USA, 635–646. https://doi.org/10.1145/2642937.2642977
[14] Kimberly Tam, Salahuddin J Khan, Aristide Fattori, and Lorenzo Cavallaro. 2015. CopperDroid: Automatic Reconstruction of Android Malware Behaviors.. In *NDSS*.
[15] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou. 2012. SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '12)*. ACM, New York, NY, USA, 93–104. https://doi.org/10.1145/2381934.2381950
[16] Wu Zhou, Zhi Wang, Yajin Zhou, and Xuxian Jiang. 2014. DIVILAR: Diversifying Intermediate Language for Anti-repackaging on Android Platform. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy (CODASPY '14)*. ACM, New York, NY, USA, 199–210. https://doi.org/10.1145/2557547.2557558