BrainTest - A New Level of Sophistication in Mobile Malware

By bferrite

Published: 2015-09-21 · Archived: 2026-04-10 02:11:12 UTC

Check Point Mobile Threat Prevention has detected two instances of a mobile malware variant infecting multiple devices within the Check Point customer base.

The malware, packaged within an Android game app called BrainTest, had been published to Google Play twice. Each instance had between 100,000 and 500,000 downloads according to Google Play statistics, reaching an aggregated infection rate of between 200,000 and 1 million users. Check Point reached out to Google on September 10, 2015, and the app containing the malware was removed from Google Play on September 15, 2015.

Overview

The malware was first detected on a Nexus 5 smartphone, and although the user attempted to remove the infected app, the malware reappeared on the same device shortly thereafter. Our analysis of the malware shows it uses multiple, advanced techniques to avoid Google Play malware detection and to maintain persistency on target devices.

Once this malware was detected on a device, Mobile Threat Prevention adjusted security policies on the Mobile Device Management solution (MobileIron) managing the affected devices automatically, thereby blocking enterprise access from the infected devices.

While the malware is capable of facilitating various cyber-criminal goals, our team confirmed it’s currently installing additional apps on infected devices. Disturbingly, the malware establishes a rootkit on the device, allowing it to download and execute any code a cybercriminal would want to run on a device. For example, it could be used to display unwanted and annoying advertisements on a device, or potentially, to download and deploy a payload that steals credentials from an infected device.

Highlights

- Samples of the malicious code found in BrainTest have been found on Google Play, and its creator has used multiple methods to evade detection by Google including
    - Bypassing Google Bouncer by detecting if the malware is being run from an IP or domain mapped to Google Bouncer and, if so, it will not perform its intended malicious activities.
    - Combining timebombs, dynamic code loading, and use of reflection to complicate reverse engineering of the malware.
    - Using off-the-shelf obfuscation (packer) from Baidu to re-introduce the malware to Google Play after the first instance was removed on Aug 24th.
- BrainTest uses four privilege escalation exploits to gain root access on a device and to install a persistent malware as a system application.

- BrainTest leverages an anti-uninstall watchdog that uses two system applications to monitor the removal of one of the components and reinstall the component.

After the the first instance of BrainTest was detected, Google removed the app from Google Play. Within days, the Check Point research team detected another instance with a different package name but which uses the same code. The malware's creators had used obfuscation to upload the new piece of malware to Google Play.

Technical Analysis

The malware consists of 2 applications:

1. **The Dropper**: Brain Test (Unpacked – com.mile.brain, Packed – com.zmhitlte.brain) This is installed from Google Play and downloads an exploit pack from the server to obtain root access on a device. If root access is obtained, the application downloads a malicious .apk file (**The Backdoor**) from the server and installs it as system application.
2. **The Backdoor**: System malware (mcpef.apk and brother.apk) This tries a few persistence methods by using few anti-uninstall techniques (described below) and downloads and executes code from server **without user consent**.

Detailed Malware Structure

- **com.mile.brain (**SHA256**: 135d6acff3ca27e6e7997429e5f8051f88215d12351e4103f8344cd66611e0f3):** This is the main application found on Google Play. It contains encrypted java archive "start.ogg" in the assets directory and dynamically loads code with dalvik.system.DexClassLoader.
- **do.jar (**SHA256**:a711e620246d9954510d3f1c8d5c784bacc78069a5c57b9ec09c3e234bc33a8b)** : The decrypted file that was created by "start.ogg." It sends a request to the server with the device's configuration. The server's response is a json, containing a link to a .jar file, class name and method name to be executed with reflection API. The application downloads the file and dynamically loads it using dalvik.system.DexClassLoader and invokes class and method specified in json.
- **jhfrte.jar:** This is a java archive file downloaded from server. If a device isn't rooted, it downloads from the server an exploit pack and executes it to obtain root on device. Once root is obtained, it downloads an additional APK file from the server (**mcpef.apk**) and installs it as system application (/system directory).
- **r1-r4:** This is a local privilege escalation (root) exploit, which includes: CVE-2013-6282, camerageroot (http://www.77169.org/exploits/2013/20130414031700), a rooting tool for mtk6592 and addtional exploit.
- **nis**: The su application used to execute shell commands with root privileges.
- **mcpef.apk** (SHA256: a8e7dfac00adf661d371ac52bddc03b543bd6b7aa41314b255e53d810931ceac): The malicious system application downloaded from server (package name – com.android.music.helper). This installs additional application from assets directory (**brother.apk**) and listens for PACKAGE_REMOVED events. If brother.apk application is removed, mcpef.apk reinstalls brother.apk from assets.
- **brother.apk** (SHA256: 422fec2e201600bb2ea3140951563f8c6fbd4f8279a04a164aca5e8e753c40e8**)** : The package name – com.android.system.certificate. System application installed by mcpef.apk. This has the same functionality as mcpef.apk. In addition, it monitors to verify if com.android.music.helper package is removed. If mcpef.apk is removed, brother.apk reinstalls it from a META-INF/brother file
- **boy, post.sh**: The shell scripts u sed for application persistency.

Application lifecycle

Google Bouncer Bypass

On start, the application checks if it is executed on one of the Google servers:

- IP ranges 209.85.128.0-209.85.255.255, 216.58.192.0-216.58.223.255, 173.194.0.0-173.194.255.255, 74.125.0.0-74.125.255.255
- or if it is executed on IP hosted domain that contains the following strings: “google”, ”android”, ”1e100”.

If any of these conditions is true, the application does not continue to execute the malicious flow. This method is design to bypass the automatic Google Play protection mechanism called Bouncer.

Timebombs, Dynamic Code Loading and Reflection

If Google Bouncer was not detected, the application starts a time bomb which initiates the malicious flow only after 20 seconds and will run every 2 hours. The time bomb triggers unpacker thread. Unpacker thread decrypt java archive from assets directory “start.ogg”, and dynamically loads it and calls the method “a.a.a.b” from this archive.

This method checks if eight hours have passed from the first run of application, and if so, request containing the device’s data to the server. The server sends back encoded json containing URL, class name and method name. Then the application downloads java archive from the URL specified in json, dynamically loads it with class loader API. Once archive is loaded, the application uses reflection api to call methods from the class names specified in the json.

Rooting and Ad Network Presentation

The reflection loaded methods check if the device is rooted. If not, the application downloads a pack of exploits from the server and runs them one-by-one up until root is achieved.

As root, the application copies su binary to /system/bin directory and silently downloads apk file from the server. Then, the APK is installed as system application and registers listener on USER_PRESENT event. This event triggers archive downloading thread. Once the event is triggered, it registers a timer. The timer triggers additional thread which makes a request to the server. It expects a json with url, class and method name. It downloads one more archive and dynamically loads code from it.

The final APK is downloaded from a different URL that is currently down, we assume that the apk purpose is overlaying ads on the screen, we assume this based on the research we have done on the API we found which returns URL of random APK file containing different advertising networks.

Persistency Watch-Dog

The application contains protection against its own removal. As outlined in the diagram above, It installs an additional application with the same functionality and these two applications monitor the removal of each other. If

one of the applications is deleted, the second application downloads and re-installs the removed one.

Network activity

BrainTest communicates with five servers:

- **APK files provider** (http://psserviceonline[.]com/): This server provides APK files with advertising network. We found two functions:
    - The first function is http://s.psserviceonline[.]com/api/s2s/tracks/ and is used for activation.
    - The second function is http://s.psserviceonline[.]com/api/ads/ which is used for obtaining a link to APK file. Regardless of the parameters, it returns a json containing a link for APK file.
- **File Server** (http://www.psservicedl[.]com): Contains android packages, java archives and zip archives with exploits
- **Archive Link domains**: Three domains with the same functionality, but the application chooses one of them to send request for archive link.
    - http://www.himobilephone[.]com
    - http://www.adsuperiorstore[.]com
    - http://www.i4vip[.]com

Counter Measures

Use an up to date anti-malware software that is capable of identifying this threat.

If the threat reappears on the device after the first installation, it means that the malware managed to install the persistency module in the System directory. In this case, the device should be re-flashed with an official ROM.
