Coordinated Disclosure of Vulnerabilities in McAfee Security Android 4.8.0.370

1 Executive summary

Researchers of MRG Effitas tested the McAfee Security Android application. During use, we came across implementation details, which might undermine the Vendor's efforts to provide a comprehensive mobile security solution with the potential to aid users in case of encounters with malware.

Testing covered the following application version.

Application name	McAfee Security
Store URL	https://play.google.com/store/apps/details?id=com.wsandroid.suite
Version	4.8.0.370

We considered the situation and opted for a responsible disclosure approach to aid the Vendor in their efforts. In accordance with industry standards, we disclose the issues based on Google's 90-day policy. As a result, after a 90-day plus a 14-day grace period, we make the discoveries public. For further details, refer to the following URL.

Disclosure date: 06 April 17
Grace period begins: 05 July 17
Grace period ends: 19 July 17

https://security.googleblog.com/2014/07/announcing-project-zero.html
https://googleprojectzero.blogspot.hu/2015/02/feedback-and-data-driven-updates-to.html

2 Vulnerability description

2.1 No user warning on rooted/compromised devices

Finding

Testing discovered that the application was normally installed and initialized on a rooted test device, provided with a dynamic hooking framework application (Xposed framework). This scenario is a highly insecure one, as the hooking framework can be utilized to bypass every security measure of the McAfee Security application. This is by no means something that an Android security suite can be properly prepared for, nevertheless no user warning has been displayed stating that the entire operation of the suite is compromised.

Risk

As a result, a highly sophisticated piece of malware using dynamic hooking might even abuse the application to convince the user that there are no threats on the device. Note that no cloaking has taken place and as per the logs, the Xposed installer application has even been scanned by the application.

Consider the following scenario.

1. The Android device with no preliminary protection, gets infected by a highly sophisticated piece of malware.
2. The users suspects that there might be something going on and opts for the suite to run a security scan on his device.
3. Upon installation and scanning, no warning is displayed and the user is reassured that his device is clean from malware.

Recommendation
It is recommended to at least warn the user in such clear cases that there might be something going on with his device and that the scan results are not trustworthy in any way.

Nevertheless, it is important to point out that reliable root detection is theoretically impossible in untrustworthy environments (such as a user controlled Android device). When it comes to root detection, false positive errors are rather uncommon, however false negatives are more dangerous from security perspective. It is possible to install readily prepared tools to fool common root detection methods (e.g. RootCloak), however, presence of the Xposed installer and the RootCloak application itself is a good indication that the device has been rooted. All in all, a comprehensive approach is recommended, where the user is notified at the slightest sign of deviation from a 'clean' device state.

As an alternate approach, it is possible to harden the application itself to detect tampering (both off-line patching plus re-packaging and runtime hooking) to some extent. Should any of the below checks fail, assume that the environment is tampered with and notify the user. Note that even a careful implementation of all below methods can only raise the bar in terms of attacker skill, motivation and expertise, it can deter and detect novice users, who solely rely on off-the-shelf tools.

- Upon startup, check if the package has debug mode enabled.
- Regularly check the signature and the signing key of the app package if it deviates from an expected value.
- List the calling stack of security related methods (such as certificate pinning, root detection etc.) Should the RootCloak tool be used, the calling stack contains elements that are clearly not from the originating application.

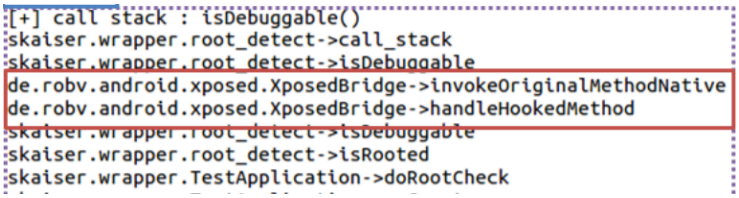

Figure 1 Evidence of dynamic hooking based tampering

For further information, refer to the following whitepaper (An Android Application Protection Scheme against Dynamic Reverse Engineering Attacks. *Lim, Yeong et al*. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, issue September 2016.)

http://isyou.info/jowua/papers/jowua-v7n3-3.pdf

2.2 No certificate pinning

Finding

Testing revealed that the application does not utilize certificate pinning. Instead, it utilizes the OS CA store.

Risk

As a result, it is trivial to bypass certificate validation using various methods (e.g consider a social engineering attack, where the user is asked to install a certificate authority on his Android device). Considering the fact that mobile security suites are intended also for non-security conscious users, this design consideration should be re-evaluated.

When a new package is installed the hash of the package is checked from a reputational perspective within the cloud service.

```
POST /aa HTTP/1.1
X-McAfee-AA-API: 2
Content-Type: text/xml
X-McAfee-MAC-Key: 688c7d38-9931-483f-a4d0-87d4f336d08b
X-McAfee-AA-APINAME: APP
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1; McAfee_511
Build/LMY47D)
Host: appcloud.mcafee.com
Connection: close
Content-Length: 839

<?xml version='1.0' encoding='UTF-8' standalone='yes' ?><request
locale="en_US"><prop name="uuid">448500e1badedf54</prop><prop
name="os">Android</prop><prop name="os_ver">5.1</prop><prop
name="client_name">com.wsandroid.suite</prop><prop
name="client_ver">4.8.0.370</prop><prop name="aff_id">0</prop><prop
name="c_code">1</prop><prop name="device_make">unknown</prop><prop
name="device_model">McAfee_511</prop><prop
name="mnc">260</prop><app><size>3618700</size><name>com.regeemir.jeopar
dissau</name><ver>136</ver><time>1969-12-31T23:59:59.999Z</time><type>2
</type><hash>086bde2ad53d7e5496892a5db930bf7975f9f213deaa9c77e1bdf2854b
5327ae</hash><devId>16a4fca37d867a79bb872294aefbe62c10516883eab6be896ad
beb996b250634</devId><dexHash
name="classes.dex">652baa505528e688cf7dd3defbd2a6bc3ca3476adbb51e1b3ace
2ddd4c4efd72</dexHash></app></request>
```

Testing revealed that an intercepting attacker or a piece of malware can avoid being detected in the cloud by changing the submitted hash values.

Note that the risk is significantly decreased by the fact that the application does not solely rely on cloud based detection methods, off-line analysis also takes place.

2.3 Traces of user activity stored in plain-text format

Finding

Testing revealed that the application maintains traces of user activity within the sandbox. For instance, the %sandbox%/databases/AP_SDK_DB_ discloses applications that have been installed since the initialisation of the McAfee suite, even if the application has been removed from the device.

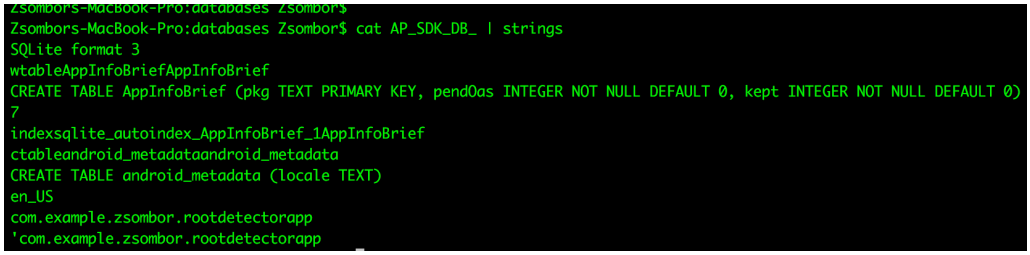

Figure 2 Traces of user activity in plain-text format

Risk

An analysis of the application sandbox reveals information regarding user activity (e.g. the names of installed applications). The risk is increased by the fact that the application maintains the log, even in case the device is rooted.

Recommendation

It is recommended to encrypt all trace of user activity, even within the sandbox. Furthermore, the user should be provided with options to delete all traces of his activity from the sandbox using the application GUI.

3 Timeline

2017.04.07 – Vendor notified, no response received
2017.05.10 – Vendor notified again
2017.05.29 – Vendor responded
2017.07.26 – Publication of the report