Uncovering the Dilemmas on Antivirus Software Design in Modern Mobile Platforms

Heqing Huang[1(✉)], Kai Chen[1,2], Peng Liu[1], Sencun Zhu[1], and Dinghao Wu[1]

[1] The Pennsylvania State University, State College, USA
{hhuang,szhu}@cse.psu.edu, chenkai010@gmail.com,
{pliu,dwu}@ist.psu.edu

[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

Abstract. With the rapid increase in Android device popularity, a new evolving arms-race is happening between the malware writers and AntiVirus Detectors (AVDs) on the popular mobile system. In its latest comparison of AVDs, independent test lab AV-TEST reported that AVDs have around 95 % malware recognition rate. However, as mobile systems are specially designed, we consider that the power of AVDs' should also be evaluated based on their runtime malware detection capabilities. In this work, we performed a comprehensive study on ten popular Android AVDs to evaluate the effectiveness of their scanning operations. During our analysis, we identified the design dilemmas related to two types of malware scanning operations, namely *local* malware scan and *cloud-based* malware scan. Our work opens a new research direction in designing more effective and efficient malware scan mechanisms for current antivirus software on mobile devices.

Keywords: Android · Antivirus · Malware · Mobile attacks

1 Introduction

The increasingly popularity of mobile computing devices (e.g., smartphones and tablets) attracts both normal users and malware writers. Among the popular mobile platforms, Android has not only conquered a lion's share of the market, but also gained the 98.1 % share of detected mobile malware in 2013 [3]. Therefore, being aware of the notorious fact of mobile malware shares, many reputable companies on PC security as well as new startups have turned their attention to mobile-platform security and released their antivirus detectors (AVDs) particularly for Android [1]. Here, an AVD generically refers to the signature-based antivirus detector that is deployed on mobile devices.

For AVDs on desktop and server systems, earlier work has studied the impact of polymorphic attacks [16] or file format confusion based attacks [15] on the malware scanning operation. Recently, the real world polymorphic attacks have also been reported [4] and further studied [22] for Android AVD evasion. In the era of mobile computing, a new evolving arms-race is going on between the malware writers and the AVDs. The AVD based on dynamic behavior or

J. Tian et al. (Eds.): SecureComm 2014, Part II, LNICST 153, pp. 359–366, 2015.
DOI: 10.1007/978-3-319-23802-9_27

other dynamic heuristics are comparatively hard to be deployed on the battery constrained mobile devices. Also, because of the centralized software distribution on Google Play, static signature based malware fingerprinting scheme gains great values, since the potential malware spreading sources are very limited (users are not suggested to install apps from untrusted sources).

Every three months, the independent AVD test lab AV-TEST generates a report [2], comparing the detection rate and usability of Android AVDs. The latest report indicates that the popular AVDs under test achieve an average detection rate of around 95 % for known malicious app samples. However, since apps are allowed to dynamically load code from external sources at runtime [21], when combined with repackaging techniques [7,14,29], malware writers demonstrates [31,32] that more advanced malware can be easily created to perform targeted attacks. Therefore, the success of the AVD's malware scan should also be measured based on its real-time detection of advanced malware.

Hence, we conduct an empirical study of ten Android AVDs on two types of malware scan operations, namely the *Local-malScan* and the *Cloud-malScan*. Our analysis result indicates that both *malScan* operations have fundamental design deficiencies. Therefore, AVD vendors should consider the design of malware scanner on Android more thoroughly.

2 Antivirus Detectors on Android Platform

In this section, we first briefly introduce some necessary background on Android Antivirus Detectors (AVDs), and then explain how we conduct the empirical study towards further understanding of the design characteristics of the current AVDs. Our analysis, particularly focuses on the real-time detection capability of the AVDs deployed on Android. More comprehensive discussions on Android security mechanisms can be found in Yan and Yin [26] and Enck et al. [11].

Table 1. Popular Antivirus Detectors (AVDs) in Our Study

ID	Vendor	AVD package name & version #	Downloads #
1	Avast	com.avast.android..........3.0.6915	50M-100M
2	AVG	com.antivirus.........................3.6	100M-150M
3	Avira	com.avira.android..................3.1	1M-5M
4	Bitdefender	com.bitdefender.security.2.8.217	1M-5M
5	Kaspersky	com.kms (premium)..........11.2.3	5M-10M
6	ESET	com.eset.ems2.gp...........2.0.843	1M-5M
7	Dr. Web	com.drweb.pro................7.00.11	10M-50M
8	Lookout	com.lookout...........8.28-879ce69	50M-100M
9	McAfee	com.wsandroid.suite.....4.0.0.143	5M-10M
10	Norton	com.symantec.mbsec.....3.8.0.12	10M-50M

Android is an operating system based on the Linux Kernel, with new features such as the Binder IPC mechanism, Power Manager and Ashmem mechanism and etc. On top of the Linux kernel, Android is loaded with four software layers, namely System Libraries, Android Runtime, Application Framework and Application. In addition to the native Linux basic discretionary access control mechanism and the SEAndroid [24] mechanism based on Linux Security Module, Android provides a fine-grained permission mechanism for all the apps running on the Application layer, including all the AVDs from third party vendors. Table 1 lists ten popular AVDs in Google Play as of Feb. 2014. The popularity of these AVDs is reflected by their overall protection rankings, according to AV Test Reports [2] for the period of Sept. 2013–Jan. 2014.

Table 2. Intents Registered and Permissions Asked by AVDs

Intents Registered	#	Permissions Requested	#
intent.action.MEDIA_REMOVED	1	android.permission.SUPERUSER	2
intent.action.MEDIA_CHECKING	3	android.permission.BATTERY_STATS	3
intent.action.PWR_DISCONNECTED	3	android.permission.google.c2dm.RECEIVE	3
intent.action.WIFI_STATE_CHANGED	4	android.permission.KILL_PROCESSES	4
intent.action.DATE_CHANGED	4	android.permission.COARSE_LOCATION	4
intent.action.SERVICE_STATE	4	android.permission.ALERT_WINDOW	5
intent.action.DIAL	5	android.permission.WRITE_BOOKMARKS	5
intent.action.MEDIA_UNMOUNTED	6	android.permission.GET_ACCOUNTS	6
intent.action.POWER_CONNECTED	6	android.permission.READ_SMS	7
intent.action.net.wifi.STATE_CHANGE	7	android.permission.READ_BOOKMARKS	7
intent.action.MEDIA_EJECT	7	android.permission.READ_CONTACTS	8
intent.action.USER_PRESENT	7	android.permission.RECEIVE_SMS	8
intent.action.ACTION_SHUTDOWN	7	android.permission.SEND_SMS	8
intent.action.NEW_OUTGOING_CALL	9	android.permission.READ_LOGS	9
intent.action.PHONE_STATE	10	android.permission.GET_TASKS	10
intent.action.PACKAGE_REPLACED	10	android.permission.WAKE_LOCK	10
intent.action.PACKAGE_REMOVED	10	android.permission.EXTERNAL_STORAGE	10
intent.action.PACKAGE_ADDED	10	android.permission.READ_PHONE_STATE	10
intent.action.BOOT_COMPLETED	10	android.permission.BOOT_COMPLETED	10

Generally, Android uses a standard template process called Zygote, which is the parent process for all the Android DVM processes, including all the AVDs' main processes. Each AVD is assigned its own unique user ID (UID) at the install time, and the access control bits for the relevant files and folders in the file system are then set accordingly by the system. The dedicated group ID (GID) numbers are assigned based on the requested permissions for the Android system resources. Also, various system daemons and apps are classified into different access control domains in the SEAndroid policy rules, in order to provide better isolation and security.

An AVD registers itself to specific broadcast intents by programmatically registering a broadcast receiver in the code or claiming the relevant receivers in the file AndroidManifest.xml. For example, an AVD may register for the system generated intents BOOT_COMPLETED, which is fired by the system once the boot process is completed. This enables the AVD to keep track of some system events of interest that are happening and then take appropriate actions.

In Table 2, the left two columns list the types of Intent actions and how frequently they are registered by the ten AVDs in our study, and the right two columns list the types of permissions and how frequently they are requested by these AVDs. From the table, it seems that the current AVDs can provide a very good real-time protection. For instance, all these AVDs listen to BOOT_COMPLETED system event to provide complete protection after the system boots up and obtain the WAKE_LOCK permission to periodically wake up the CPU to keep monitoring the system status. Also, events like PACKAGE_ADDED and PACKAGE_REMOVED are mostly registered to help monitor the newly installed or updated Android application package (APK) files.

3 Dilemmas for Malware Scan Design

3.1 Local Malware Scan Dilemma

Scan the Archived Files or Not? Our study shows that current Android AVDs have designed a comprehensive local malware scan (local-malScan), which is a thorough scan carried out on the pre-selected (sub)directories, which usually includes operations like file preprocessing and malware signature fingerprinting. Due to the power or other resource constraints, the local-malScan usually does not perform thorough file preprocessing on the files with specific formats (e.g., the archived files). Therefore, the malicious payload can be simply zipped and dropped on the file system without being identified. While some AVDs perform comprehensive scan by uncompressing the archived format files, we discover that one can construct a multi-layered archive file to conduct denial of service attacks and drain the device battery, since the scanner will keep unzipping every inner zip file in the multi-layered archive file diligently. As such, whether to preprocess or scan the special formatted files is a dilemma on current resource restricted AVD on mobile platforms.

Update the Virus Definition File or Not? During our comprehensive analysis, we discover an interesting *probing channel*. Almost all the AVDs will have the VDF file and other permanent data or cached files stored in the subdirectory at /data/[AVD_package]/*. These files are set to be "world unreadable" and enforced by Linux kernel in Android using access control policies. We find that this solid design of app data privacy protection is not enough for AVD deployment, since an adversary only needs to know the file sizes or other meta-data information of relevant files (e.g., creation and update time) in the subdirectory to infer the updating status of these files. During our analysis, we discover that by using the /system/bin/ls program, or writing a dynamic library which calls the stat() system call, one can directly probe the meta-data information of all these "world unreadable" files in an AVD's data folder. This design deficiency can potentially be leveraged to design on-demand malware polymorphism. Basically, whenever the anti-AVD app detects a VDF-update, which might contain the signature to fingerprint its current malicious payloads, it can update its payloads using a new polymorphic strategy. Therefore, the adversary will enjoy this on-demand VDF-update feature and is always one-step ahead of the AVD's static fingerprinting. Hence, whether to perform the VDF update is another dilemma for current AVD Local-malScan.

3.2 Cloud Malware Scan Dilemma

To Offload or Not to Offload? Due to the limitation of the on the Local-malScan, we sense a trend of adding the cloud-based scanning strategy for mobile platform during our analysis. Cloud-based scan (Cloud-malScan) is generally believed to be suitable for resource limited mobile devices, as it can offload the heavy computation to a remote server by sending out the collected information, including the file hashing value, the meta-data of a file etc. However, since the per UID network usage statistics can also be probed in "/proc/uid_stat/[AVD_uid]/snd(rcv)" an adversary can plan evasions against AVDs by identifying the network sending and receiving statistic pattern of Cloud-malScan. So we find that the implicit dilemma in the Cloud-malScan is whether to send enough file information to the remote server for further signature mapping and scanning. If the Cloud-malScan on the local device tries to collect less information to send out (e.g., only the file hash value or the file meta-data), then the malware scan/detection performed on the server can merely based on simple signature fingerprinting. However, if the Cloud-malScan collects more information (e.g., execution traces) to offload to remote server for deep (behavior based) analysis, its network statistics become more identifiable and is vulnerable to targeted evasions (e.g., the malicious payload will be loaded only after the Cloud-malScan performed on the device).

4 Related Work

Antivirus evasion techniques [5,15,16] have been studied previously. Oberheide et al. [17] has also generally discussed challenges in deploying antivirus detectors (AVDs) on mobile platforms. Android Dalvik Bytecode polymorphic transformation attacks have been presented by Rastogi et al. [22]. Our new evasion techniques exploit the cloud-based malware scanning behavior of the AVDs, and they are complementary to obfuscation-based or other evasion techniques, which are similar to the other attacks discussed in [12].

Malware and intrusion analysis techniques [10,11,13,26] have been designed and applied for offline analysis. Also, various interesting anti-analysis technique have been discussed [9,19] for malware on both mobile and PC. Our anti-AVD app design is conceptually similar to anti-analysis techniques, but we emphasize more on the evasion of AVD's online protection mechanism. Zhou et al. [32] provide a survey of Android malware, and similarly, the discovered design dilemmas in this paper are based on a systematic survey of ten popular Android AVDs.

Side/timing channel issue [8,18,23,25,30] is also an active research aspect in both mobile and PC era. The network based probing and fingerprinting based attacks for the AVD deployed on the mail server side have been explored by Oberheide et al. [18], also including the reconnaissance and action phases. Information hiding techniques have been discussed by Petitcolas et al. [20]. Side channel/timing channel preventions have been discussed in several papers [6,27,28]. Generally, it is one of the toughest challenges in computer security. Zhang et al. [28] provides the language-based control and mitigation for the timing channels.

5 Conclusion

Through an empirical study of ten top AVDs on the current Android platform, we identified several design dilemmas in the malware scan operations, including the local malware scan and the cloud-based malware scan. These dilemmas are related to the malware scan of the archived or other special formatted file, the virus definition file update, and the offloading file sizes of the cloud-based malware scan, and pose challenges in antivirus software design in the current Android platform. Through this study, we open a new research topic on how to improve the effectiveness and efficiency of current malware scan and detection on current mobile platforms.

References

1. Android antivirus companies. Technical report. http://www.zdnet.com/android-antivirus-comparison-review-malware-symantec-mcafee-kaspersky-sophos-norton-7000019189/
2. AV TEST report, January 2014. http://www.av-test.org/en/tests/mobile-devices/android/jan-2014/
3. Kaspersky Lab Reports Mobile Malware in 2013. http://usa.kaspersky.com/about-us/press-center/press-releases/kaspersky-lab-reports-mobile-malware-2013-more-doubles-previous
4. Server-side Polymorphic Android Applications. http://www.symantec.com/connect/blogs/server-side-polymorphic-android-applications
5. Al-Saleh, M.I., Crandall, J.R.: Application-level reconnaissance: timing channel attacks against antivirus software. In: 4th USENIX Workshop on LEET 2011 (2011)
6. Askarov, A., Zhang, D., Myers, A.C.: Predictive black-box mitigation of timing channels. In: ACM CCS 2010 (2010)
7. Chen, K., Liu, P., Zhang, Y.: Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In: ICSE, pp. 175–186 (2014)
8. Chen, S., Wang, R., Wang, X., Zhang, K.: Side-channel leaks in web applications: a reality today, a challenge tomorrow. In: S&P 2010 (2010)

9. Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J.: Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In: DSN 2008 (2008)
10. Christodorescu, M., Jha, S., Seshia, S.A., Song, D., Bryant, R.E.: Semantics-aware malware detection. In: 2005 IEEE Symposium on Security and Privacy. IEEE (2005)
11. Enck, W., Gilbert, P., Chun, B.-G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: OSDI, vol. 10, pp. 1–6 (2010)
12. Huang, H., Chen, K., Ren, C., Liu, P., Zhu, S., Wu, D.: Towards discovering and understanding unexpected hazards in tailoring antivirus software for android. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS 2015 (2015)
13. Huang, H., Zhang, S., Ou, X., Prakash, A., Sakallah, K.: Distilling critical attack graph surface iteratively through minimum-cost sat solving. In: Proceedings of the 27th Annual Computer Security Applications Conference, pp. 31–40. ACM (2011)
14. Huang, H., Zhu, S., Liu, P., Wu, D.: A framework for evaluating mobile App repackaging detection algorithms. In: Huth, M., Asokan, N., Čapkun, S., Flechais, I., Coles-Kemp, L. (eds.) TRUST 2013. LNCS, vol. 7904, pp. 169–186. Springer, Heidelberg (2013)
15. Jana, S., Shmatikov, V.: Abusing file processing in malware detectors for fun and profit. In: 2012 IEEE Symposium on Security and Privacy (SP), pp. 80–94. IEEE (2012)
16. Oberheide, J., Bailey, M., Jahanian, F.: PolyPack: an automated online packing service for optimal antivirus evasion. In: 3rd USENIX on Offensive Technologies
17. Oberheide, J., Jahanian, F.: When mobile is harder than fixed (andvice versa): demystifying security challenges in mobile environments. In: HotMobile 2010. ACM (2010)
18. Oberheide, J., Jahanian, F.: Remote fingerprinting and exploitation of mail server antivirus engines (2009)
19. Pék, G., Bencsáth, B., Buttyán, L.: nEther: in-guest detection of out-of-the-guest malware analyzers. In: Proceedings of the Fourth European Workshop on System Security, EUROSEC 2011. ACM (2011)
20. Petitcolas, F.A., Anderson, R.J., Kuhn, M.G.: Information hiding-a survey. In: Proceedings of the IEEE (1999)
21. Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., Vigna, G.: Execute this! analyzing unsafe and malicious dynamic code loading in android applications (2014)
22. Rastogi, V., Chen, Y., Jiang, X.: Droidchameleon: evaluatingandroid anti-malware against transformation attacks. In: asiaCCS. ACM (2013)
23. Schlegel, R., Zhang, K., Zhou, X.-Y., Intwala, M., Kapadia, A., Wang, X.: Soundcomber: a stealthy and context-aware sound trojan for smartphones. In: NDSS (2011)
24. Smalley, S., Craig, R.: Security enhanced (se) android: bringing flexible mac to android. In: NDSS (2013)
25. Studer, A., Passaro, T., Bauer, L.: Don't bump, shake on it: the exploitation of a popular accelerometer-based smart phone exchange and its secure replacement. In: ACSAC 2011 (2011)
26. Yan, L.K., Yin, H.: Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In: USENIX rSecurity 2012 (2012)
27. Zhang, D., Askarov, A., Myers, A.C.: Predictive mitigation of timing channels in interactive systems. In: ACM CCS, pp. 563–574. ACM (2011)

28. Zhang, D., Askarov, A., Myers, A.C.: Language-based control and mitigation of timing channels. In: ACM SIGPLAN Notices, vol. 47, pp. 99–110. ACM (2012)
29. Zhang, F., Huang, H., Zhu, S., Wu, D., Liu, P.: View-droid: towards obfuscation-resilient mobile application repackaging detection. In: Proceedings of the 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (2014)
30. Zhou, X., Demetriou, S., He, D., Naveed, M., Pan, X., Wang, X., Gunter, C.A., Nahrstedt, K.: Identity, location, disease and more: inferring your secrets from android public resources. In: ACM CCS. ACM (2013)
31. Zhou, Y., Jiang, X.: An analysis of the anserverbot trojan. http://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBotAnalysis.pdf
32. Zhou, Y., Jiang, X.: Dissecting android malware: characterizationand evolution. In: SP 2012. IEEE (2012)