Journal of Communications and Information Networks
Vol.1, No.4, Dec. 2016
DOI: 10.11959/j.issn.2096-1081.2016.052

Review paper

A survey of privacy protection techniques for mobile devices

ZHANG Lei[1], ZHU Donglai[1], YANG Zhemin[1], SUN Limin[2], YANG Min[1]

1. Fudan University, ShangHai 200433, China
2. Institute of Information Engineering, China Academy of Science, Beijing 100093, China

Abstract: Modern mobile devices provide a wide variety of services. Users are able to access these services for many sensitive tasks relating to their everyday lives (e.g., finance, home, or contacts). However, these services also provide new attack surfaces to attackers. Many efforts have been devoted to protecting mobile users from privacy leakage. In this work, we study state-of-the-art techniques for the detection and protection of privacy leakage and discuss the evolving trends of privacy research.

Key words: mobile security, privacy leakage, privacy protection, static analysis, dynamic analysis

Citation: ZHANG L, ZHU D L, YANG Z M, et al. A survey of privacy protection techniques for mobile devices[J]. Journal of communications and information networks, 2016, 1(4): 86-92.

1 Introduction

Millions of mobile applications (apps for short) are deployed for purposes ranging from gaming and social networking to more serious uses such as healthcare, finance, and even home security. As a result, mobile devices are accessed by users for many sensitive reasons, and these make them lucrative targets for cybercrimes. Users typically grant access to these data to mobile applications, without a full appreciation of the security of the information that they are exposing to a variety of third parties. As a result[1], the adversary could gain access to details such as the mobile user's bank accounts, identities, and medical data. For example, prior research[2-4] shows that a large number of Android apps fail to perform any certificate verification during an SSL connection, and some even utilize hardcoded security credentials, which provide the adversary with the ability to steal mobile users' privacy data during Internet communications.

Nevertheless, studies show that users remain concerned about their privacy and many application markets apply privacy leakage detection and protection technologies to protect users' privacy. In this work, we study the state-of-the-art techniques for detection and protection of privacy leakage and discuss the evolutionary trends of privacy research.

2 Privacy leakage detection

State-of-the-art leakage detection approaches depend on either static data propagation analysis or dynamic taint tracking. Researchers mainly focus on the propagation

Manuscript received Sept.14, 2016; accepted Nov.23, 2016
This work is supported by the Science and Technology Commission of Shanghai Municipality (No. 15511103003), the National Natural Science Foundation of China (No. 61602121), and the Open Project of Beijing Key Laboratory of IoT Information Security Technology (No. J6V0011104).

of users' sensitive data. Data propagation analysis first marks the user privacy data in apps (known as sources), and labels the channels that can be used to transmit data outside of the app (known as sinks), for example, SMS and Internet. Then, data propagation analysis identifies privacy leakage if a data propagation path exists between the source and the sink, by analyzing the source code with the call graph and control flow graph of the application. On the other hand, in dynamic taint tracking, a flag is bound to each privacy data, and represents whether the corresponding data contain privacy information. Taint tracking techniques dynamically propagate the taint information and check whether tainted data are transmitted outside of the app. The remainder of this section presents a review of research work related to privacy leakage detection.

2.1 Static data propagation analysis

Androidleaks[5] proposes a static approach to automatically checking the leakage of sensitive information in Android applications on a large scale. Comdroid[6] and Chex[7] use static analysis to detect application and discover vulnerabilities that can be used to leak user data. LeakMiner[8] uses static taint analysis for application vetting. Furthermore, FlowDroid[9] proposes a precise context, flow, field, object-sensitive and lifecycle-aware static taint analysis for Android apps. Commonly, static analysis only requires a small amount of time to analyze the whole application and gain a high coverage of the program code. However, a static analysis does not provide runtime information, thereby causing false positives/negatives. Besides, attackers can easily evade static analysis by using detection-evasion technologies, such as code obfuscation.

2.2 Dynamic taint tracking

Contrary to static data propagation analysis, dynamic taint tracking detects privacy leakage by monitoring the runtime propagation flow of sensitive information. The most well-known tracker is TaintDroid[10], which modifies the code in the Android Dalvik VM. TaintDroid uses instrumentation to dynamically monitor data propagation in the application and warns the user when the target application transmits sensitive information to the network. Aurasium[11] uses another approach to monitor the runtime data propagation of the application, by injecting monitoring code and repacking the application. Compared to static data propagation analysis, dynamic taint tracking can achieve a higher precision for identifying privacy leakage, but it is hard to gain satisfactory coverage of the application code. However, when applied to large-scale analysis, dynamic taint tracking is time-consuming because it needs to be performed while the app is running.

2.3 Combine static data propagation analysis and dynamic taint tracking

Researchers have proposed exploiting the advantages of both static data propagation analysis and dynamic taint tracking by presenting approaches that combine static and dynamic analysis. For example, AppIntent[12] first uses static data propagation analysis to generate a set of candidates possibly responsible for leaking user data and then uses symbolic execution to remove infeasible data propagation paths. In this way, AppIntent[12] can increase its precision without losing its coverage of application code, and can detect the privacy leakage with low time cost. AspectDroid[13] uses static bytecode instrumentation to weave monitoring code into an existing app and then executes the repackaged apps to investigate Android apps for possible unwanted behavior. However, this kind of hybrid approach is prone to detection-evasion attacks used for either static data propagation analysis or dynamic taint tracking.

3 Privacy protection

The common way to protect user data from privacy leakage is to prohibit the unsafe use of sensitive information. Based on this idea, the following approaches are introduced to limit the abuse of system resources.

3.1 Permission and access control

Both Android and iOS have run-time permission models that require users to explicitly approve a request by an application to access private data or potentially dangerous functionality. However, mobile users often do not understand the permission descriptions and there is no way of granting some permissions and denying others. Apex[14] proposes an approach that the mobile user can grant permissions to apps or the accesses to system resources on demand. DescribeME[15] proposes a novel technique to automatically generate security-centric app descriptions, which are readable and help users avoid privacy-breaching apps. AppFence[16] presents two privacy controls to empower users to protect their data from exfiltration by permission-hungry applications and prevent the mobile application from collecting user data based on dynamically monitoring application behaviors. FlaskDroid[17] provides simultaneous mandatory access control on both the middleware and kernel layers of the Android OS. However, this kind of approach is unable to decrease the false positive rate and they commonly require the implementation of security policies that mobile users and developers find hard to understand.

3.2 Sandboxing and isolation

The sandboxing capabilities of the Android OS centered on the user separation capabilities of Linux. Researchers have enhanced the Android sandboxing mechanism to protect apps when they interact with the user's privacy information. For example, Aurasium[11] proposes a user-level sandboxing and repackages Android applications to monitor the behaviors of retrieving user's sensitive information. SplitDroid[18] splits the execution of an app into two components, in which the sensitive component is isolated from the normal component and the sensitive data can not be accessed by the normal component. On the other hand, AFrame[19], AdDroid[20], AdSplit[21] propose approaches to isolate untrusted processes that utilize sensitive system sources.

3.3 Security property

Researchers have attempted to enforce the protection of mobile users' privacy by identifying rules of safe programming practice. They write down these rules as security properties, and check whether these properties are followed by application developers. Kirin[22] designs security rules, which can be used to match unwanted properties in the security configuration of Android applications. Saint[23] uses security policies to manage the grant of permissions at install-time and run-time. Mops[24] defines the security property as a finite state automaton, and identify whether any state of the application violate the pre-defined security property.

4 New problems in recent years

Considering the huge number of different apps and various kinds of leakage channels, it is hard to provide comprehensive protection for mobile users. Additionally, increasing kinds of user data have to be treated as private sources and the judgment used to identify privacy leakage should be amended. These are challenges for the technologies used for detecting privacy leakages. In the following, we introduce three new areas of research focus.

4.1 User intent

Traditionally, the detection of privacy leakage on mobiles focus on the transmission of sensitive data[10,25]. They commonly identify privacy leakages by checking whether sensitive data leaves the device. However, many mobile apps use cloud services to store their data, thus they need to transmit user information (sensitive or not) to remote data server. As a result, recognizing sensitive data transmission as privacy leakage is not precise. Actually, we found that a lot of benign apps collect and send user sensitive data (for example, contacts, location, etc.) to their cloud servers. Since this kind of data transmission is notified to the mobile user, it should not be treated as privacy leakage. Therefore, the transmission of sensitive data by itself may not indicate privacy leakage. Thus, AppIntent[12] claims that a more appropriate indicator should be whether the transmission is user-intended:

1) User-intended data transmission. Some data transmissions are granted by the mobile users, since they are required to fulfill some ongoing user operation. For example, an SMS management[26] app can allow the user to forward an SMS message on demand, by clicking several buttons on the mobile screen. Another example is the Map application. When using location-based services[27], the mobile user knows and allows that the location data will be sent to the internet for retrieving location-based web contents. Therefore, this kind of data transmission should not be treated as privacy leakage.

2) Unintended data transmission. This kind of data transmission is conducted stealthily, and should be recognized as privacy leakage. Commonly, it does not need any user operation on user-interfaces, such as clicking a button, inputing text. As a result, the mobile user is not aware of the data transmission.

Other than AppIntent[12], other researches also contribute on the verification of privacy leakage. Gilbert, et al.[28] use an EULA (End-User-License Agreement) to explicitly notify the user of the sensitive data usage at run-time. Then, the user can decide whether this usage should be granted. Similarly, BLADE[29] recognizes web malware by identifying whether an explicit notification is presented. Additionally, Pegasus[30] proposes an approach which detects the malicious behaviors of applications. It characterizes the app behaviors based on the temporal sequence when the apps use critical APIs and permissions, and verifies malicious behaviors which are inconsistent with the user-interface operations. However, privacy leakage cannot be simply modeled as the usage sequence of permissions or APIs. Thus, Pegasus missed many kinds of privacy leakage. Besides, VetDroid[31] utilizes a dynamic taint tracking by generating specifications for sensitive operations. However, it mainly focuses on the application logic rather than the triggering condition of each operation.

4.2 Privacy source identification

Identifying sensitive user input is a prerequisite for privacy protection. Traditional work mainly focuses on identify sensitive data acquired through well-defined system APIs. For example, PScout[32] implements a static analysis tool for Android source code and generates a complete permission-to-API mapping by a version-independent analysis. SUSI[33] proposes a machine-learning approach and categorizes additional sources and sinks which are not included in previous research. However, this kind of work only label those well-defined data, but user input data through an UI (User Interface) also faces the threats of privacy leakage.

As reported in Refs.[34-36], the adversaries can steal sensitive user input by exploiting the vulnerabilities in target system. For example, malicious apps use very similar UIs to steal users' bank accounts and passwords. Additionally, benign apps do not encrypt their sensitive data and write

plaintext contents into files which are not well-protected. Research work in side channels[37] and content-pollution vulnerabilities[38] also proved that adversary can stealthily obtain sensitive user input.

Given its importance, UIpicker[39] claims that user-inputted privacy data needs protection urgently. Unlike those well-defined data which can be automatically labeled by analyzing critical APIs, sensitive data from user inputs cannot be recognized if we do not interpret the context and semantics of apps' UIs. To solve this problem, prior work[10, 40] propose approaches which rely on manually labeling of the input contents which needed to be protected. This is inconvenient and ineffective to protect user privacy in a large amounts of apps. Another approach[10] is to label all user inputs as sensitive and should be protected, which is much more imprecise and introduces lots of false positives.

With an observation that most privacy-related UI elements are well-described in layout resource files or annotated by relevant keywords on UI screens, UIPicker[39] and SUPOR[41] respectively proposes an approach for automatically labeling sensitive user input data, by combining of several natural language processing, machine-learning, and program analysis techniques.

4.3 Hardware-associated approach and trust zone

Some researchers leverage hardware-based features for protecting the privacy and security of sensitive data. Usually, they implement a TEE (Trusted Execution Environment) by proposing a hardware-enforced isolated execution environment for security-critical code, which can provide a safe haven for storing and processing sensitive data. The general concept behind TEEs is that this "trusted," or "secure," world, can be verified using techniques such as secure or authenticated boot[42] and can be leveraged to maintain a root of trust on the device, even when the "normal," or "non-secure," world is compromised. The most popular of these approaches, based on their market dominance, is ARM's TrustZone[43,44].

As the smartphone continues gaining popularity, many web services are using OTPs (One-Time Passwords) to verify the authentication of mobile users. As a convenient solution, mobile devices can install software-based OTP generators as software apps, and use these OTPs to communicate with remote web services. Commonly, this kind of software-based OTPs do not introduce additional burden to mobile device. However, they cannot ensure the confidentiality of generated passwords, as the OTP software can be attacked by the adversary. Moreover, if the mobile OS (Operation System) crashes, the software-based OTP will not work. Hardware-based OTPs can solve these security problems, but it may need the mobile user to carry a physical token for generating secure OTPs, which will be much more inconvenient when there are several tokens needed.

In order to combine the flexibility of software OTPs and the security of hardware OTPs, TrustOTP[44] proposes an OTP solution based on ARM TrustZone. Because of the using of ARM TrustZone technology, mobile users do not need to carry additional physical tokens. Since TrustOTP is a hardware-based solution, it does not suffer from the attacks which are commonly in the mobile OS and can work correctly when the mobile OS crashes. TrustOTP argues that it is convenient to configure multiple OTP algorithms and instances for each application and do not need to modify the mobile OS.

5 Conclusion

In this work, we compared and analyzed prior research relating to privacy leakage and protection. The numerous parties corresponding to the users' privacy data complicate building a comprehensive mobile protection system. In our view, the current

approach, wherein each component attempts to manage its own privacy protections, cannot ensure complete protection of user privacy. Besides, most of the approaches are heavy-weighted, causing poor performance when multiple privacy protection modules are enabled. Thus, we believe the effectiveness as well as the performance should be re-evaluated, to establish a comprehensive solution for privacy protection. Furthermore, we have proposed several potential research directions, as well as the major challenges.

1) The principle for judging privacy leakage can be further amended.

2) New privacy source. With the great popularity of mobile devices, excluding those discussed in this paper, additional types of user data should be labeled as sensitive information, such as pictures and videos.

3) New technologies for large-scale analysis. It remains difficult to conduct a fast, large-scale, and precise analysis for detecting privacy leakage in Android applications. Besides, only a few existing approaches can accommodate detection-evasion technologies.

References :

[1] PCWorld. Skype for android has a nasty vulnerability[EB/OL]. http://www.pcworld.com/article/225301/skype_for_android_has_a_nasty_vulnerability.html.

[2] CHIN E, WAGNER D. Bifocals: analyzing webview vulnerabilities in android applications[M]//Information Security Applications. Springer International Publishing, 2013:138-159.

[3] EGELE M, BRUMLEY D, FRATANTONIO Y, et al. An empirical study of cryptographic misuse in android applications[C]//ACM Sigsac Conference on Computer & Communications Security. Berlin, Germany, 2013: 73-84.

[4] ENGLER D, CHELF B, CHOU A, et al. Checking system rules using system-specific, programmer-written compiler extensions[C]// Conference on Symposium on Operating System Design & Implementation. USENIX Association, San Diego, USA, 2000: 1-1.

[5] GIBLER C, CRUSSELL J, ERICKSON J, et al. AndroidLeaks: automatically detecting potential privacy leaks in Android applications on a large scale in trust and trustworthy computing[M]. Springer Berlin Heidelberg, 2012: 291-307.

[6] CHIN E, FELT AP, GREENWOOD K, et al. Analyzing inter-application communication in Android[C]//International Conference on Mobile Systems, Applications, and Services, Bethesda, USA, 2011: 239-252.

[7] LU L, LI Z, WU Z, et al. CHEX: statically vetting Android apps for component hijacking vulnerabilities[C]//ACM Conference on Computer and Communications Security. Raleigh, USA, 2012: 229-240.

[8] YANG Z M, YANG M. LeakMiner: detect information leakage on Android with static taint analysis[C]//The 3rd World Congress on Software Engineering, Wuhan, China, 2012: 101-104.

[9] ARZT S, RASTHOFER S, FRITZ C, et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware Taint analysis for Android Apps[J]. ACM sigplan notices, 2014, 49(6): 259-269.

[10] ENCK W, GILBERT P, HAN S, et al. TaintDroid: an information-flow tracking system for real-time privacy monitoring on smartphones[J]. ACM transactions on computer systems, 2014, 32(2): 393-407.

[11] XU R, SAÏDI H, ANDERSON R. Aurasium: practical policy enforcement for Android applications[C]//The 21st USENIX Conference on Security Symposium. Bellevue, USA, 2012: 27-27.

[12] YANG Z, YANG M, ZHANG Y, et al. AppIntent: analyzing sensitive data transmission in android for privacy leakage detection[C]//ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 2013: 1043-1054.

[13] ALI-GOMBE A, AHMED I, RICHARD III G G, et al. AspectDroid: Android App analysis system[C]//Proceedings of the 6th ACM Conference on Data and Application Security and Privacy, New Orleans, USA, 2016: 145-147.

[14] NAUMAN M, KHAN S, ZHANG X. Apex: extending android permission model and enforcement with user-defined runtime constraints[C]//ACM Symposium on Information, Computer and Communications Security, Beijing, China, 2010: 328-332.

[15] ZHANG M, DUAN Y, FENG Q, et al. Towards automatic generation of security-centric descriptions for Android Apps[C]//The 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, USA, 2015: 518-529.

[16] HORNYACK P, HAN S, JUNG J, et al. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications[C]//ACM Conference on Computer and Communications Security, Chicago, USA, 2011: 639-652.

[17] BUGIEL S, HEUSER S, SADEGHI A R. Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies[C]//Usenix Conference on Security, Washington, USA, 2013: 131-146.

[18] YAN L, GUO Y, CHEN X. SplitDroid: isolated execution of sensitive components for mobile applications in security and privacy in communication networks[M]. Springer International Publishing, 2015.

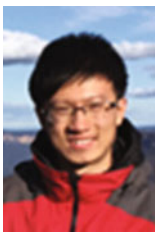
[19] ZHANG X, AHLAWAT A, DU W. AFrame: isolating advertisements from mobile applications in Android[C]//Computer Security Applications Conference, New Orleans, USA, 2013:9-18.

[20] PEARCE P, FELT A P, NUNEZ G, et al. AdDroid: privilege separation for applications and advertisers in Android[C]//The 7th

ACM Symposium on Information, Computer and Communications Security, Seoul, Korea, 2012: 71-72.
[21] SHEKHAR S, DIETZ M, WALLACH D S. AdSplit: separating smartphone advertising from applications[J]. Dissertations & theses - gradworks, 2012, 54(1): 99.
[22] ENCK W, ONGTANG M, MCDANIEL P. On lightweight mobile phone application certification[C]//ACM Conference on Computer and Communications Security, CCS 2009, Chicago, USA, 2009: 235-245.
[23] ONGTANG M, MCLAUGHLIN S, ENCK W, et al. Semantically Rich application-centric security in Android[J]. Security & communication networks, 2009, 5(6): 658-673.
[24] HAO Chen, WAGNER D. MOPS: an infrastructure for examining security properties of software[C]//Acm Conference on Computer & Communications Security. Washington, USA, 2002: 235--244.
[25] EGELE M, KRUEGEL C, KIRDA E, et al. PiOS: detecting privacy leaks in iOS applications[C]//Network and Distributed System Security Symposium, San Diego, USA, 2011: 280-291.
[26] Anzhuoduanxin[EB/OL]. http://lib.91.com/news/07302012/190845592.shtml.
[27] Google map[EB/OL]. http://www.google.com/mobile/maps/.
[28] GILBERT P, CHUN B G, COX L P, et al. Vision: automated security validation of mobile apps at app markets[C]// International Workshop on Mobile Cloud Computing and Services, Bethesda, USA, 2011:21--26.
[29] LU L, YEGNESWARAN V, PORRAS P, et al. BLADE: an attack-agnostic approach for preventing drive-by malware infections[C]// ACM Conference on Computer and Communications Security, Chicago, USA, 2010: 440-450.
[30] CHEN K Z, JOHNSON N M, D'SILVA V, et al. Contextual policy enforcement in android applications with permission event graphs[C]//Symposium on Network and Distributed System Security (NDSS), 2013.
[31] ZHANG Y, YANG M, XU B, et al. Vetting undesirable behaviors in android apps with permission use analysis[C]//The ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 2013, 9: 611-622.
[32] AU K W Y, ZHOU Y F, HUANG Z, et al. PScout: analyzing the Android permission specification[C]// Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, USA, 2012: 217-228.
[33] RASTHOFER S, STEVEN A, BODDEN E. A machine-learning approach for classifying and categorizing android sources and sinks[C]//Network and Distributed System Security Symposium, San Diego, USA, 2014.
[34] Bank app users warned over android security[EB/OL]. http://www.itpro.co.uk/android/19332/mwc-2013-bank-app-users-warned-over-android-security.
[35] Phishing attack replaces android banking apps with malware [EB/OL]. https://securingtomorrow.mcafee.com/mcafee-labs/phishing-attack-replaces-android-banking-apps-with-malware/.
[36] Av-comparatives : mobile security review-september 2014[EB/OL]. http://www.av-comparatives.org/wp-content/uploads/2014/09/avc_mob_201407_en.pdf.
[37] CHEN Q A, QIAN Z Y, MAO Z M. Peeking into your app without actually seeing it: UI state inference and novel android attacks[C]// The 23rd USENIX Conference on Security Symposium, San Diego, USA, 2014: 1037-1052.
[38] ZHOU Y J, JIANG X X. Detecting passive content leaks and pollution in android applications[C]//The 20th Network and Distributed System Security Symposium (NDSS). 2013.
[39] NAN Y H, YANG M, YANG Z M, et al. UIPicker: user-input privacy identification in mobile applications[C]//Usenix Conference on Security Symposium, Washington, USA, 2015: 993-1008.
[40] ZHOU Y J, SINGH K, JIANG X X. Owner-Centric Protection of Unstructured Data on Smartphones[M]. Trust and Trustworthy Computing. 2014: 55-73.
[41] HUANG J J, LI Z C, XIAO X S, et al. SUPOR: precise and scalable sensitive user input detection for android apps[C]// Usenix Security Symposium, Washington, USA, 2015.
[42] EKBERG J E, KOSTIAINEN K, ASOKAN N. Trusted execution environments on mobile devices[C]//ACM Sigsac Conference on Computer & Communications Security. Berlin, Germany, 2013: 1497-1498.
[43] BRASSER F, KIM D, LIEBCHEN C, et al. Regulating ARM TrustZone devices in restricted spaces[C]//The 14th Annual International Conference on Mobile Systems, Applications, and Services, Singapore, Singapore, 2016: 413-425.
[44] SUN H, SUN K, WANG Y W, et al. TrustOTP: transforming smartphones into secure one-time password tokens[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, USA, 2015: 976-988.

About the authors

ZHANG Lei was born in Henan Province. He received the B.E. degree in electronic engineering from Fudan University, Shanghai, China. He is now a Ph.D. candidate of the science and technology of computer, from Fudan University. His research interests include system security and privacy leakage. (Email: lei_zhang14@fudan.edu.cn)



YANG Zhemin [corresponding author] is a Lecturer with Software School, Fudan University, Shanghai, China. He received the B.Sc. and Ph.D. degrees in computer science from Fudan University, in 2007 and 2012, respectively. His research interests are in system security and program analysis techniques. (Email: yangzhemin@fudan.edu.cn)