Global Journal of Computer Science and Technology
Network, Web & Security
Volume 13 Issue 12 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Evaluating Smartphone Application Security: A Case Study on Android

By Muneer Ahmad Dar & Javed Parvez
University of Kashmir, India

Abstract - Currently, smart phones are becoming indispensable for meeting the social expectation ofalways staying connected and the need for an increase inproductivity are the reasons for the increase in smartphone usage. One of the leaders of the smart phone evolution is Google's Android operating system. It ishighly likely that Android is going to be installed in manymillions of cell phones during the near future. With thepopularity of Android smart phones everyone finds it convenient to make transactions through these smartphones because of the openness of Android applications. The malware attacks are also significant. Androidsecurity is complex and we evaluate an applicationdevelopment environment which is susceptible tomalware attacks. This paper evaluates Android securitywith the purpose of identifying a secure applicationdevelopment environment for performing securetransactions on Android-based smart phones.

Keywords : smartphone, android, malware, spam, vulnerabilities, attacks, mobility, API.

GJCST-E Classification : D.4.6

EVALUATING SMARTPHONE APPLICATION SECURITY A CASE STUDY ON ANDROID

Evaluating Smartphone Application Security: A Case Study on Android

Muneer Ahmad Dar [α] & Javed Parvez [σ]

Abstract - Currently, smart phones are becoming indispensable for meeting the social expectation of always staying connected and the need for an increase in productivity are the reasons for the increase in smart phone usage. One of the leaders of the smart phone evolution is Google's Android operating system. It is highly likely that Android is going to be installed in many millions of cell phones during the near future. With the popularity of Android smart phones everyone finds it convenient to make transactions through these smart phones because of the openness of Android applications. The malware attacks are also significant. Android security is complex and we evaluate an application development environment which is susceptible to malware attacks. This paper evaluates Android security with the purpose of identifying a secure application development environment for performing secure transactions on Android-based smart phones.

Keywords : smartphone, android, malware, spam, vulnerabilities, attacks, mobility, API.

I. Introduction

Smartphones have become indispensable part of our daily lives in recent years, since they are involved in keeping in touch with friends and family, doing business, accessing the internet and other activities. Andy Rubin, Google's director of mobile platforms, has commented: "There should be nothing that users can access on their desktop that they can't 0access on their cell phone" [1]. Growth in smartphone sales is depicted in the figure below.

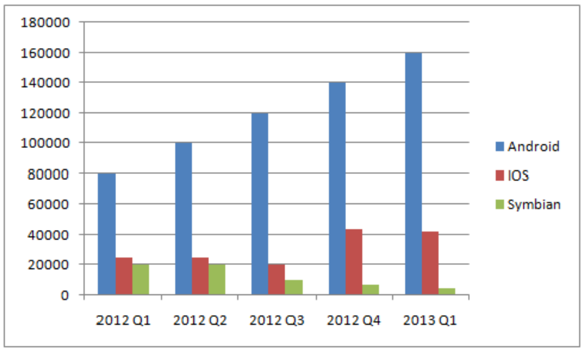

Figure 1 : Worldwide Smartphone Sales

*Author α : Scientist-B at National Institute of Electronics and Information Technology Srinagar, India, received Masters degree in Computer Applications from the University of Kashmir.
E-mail : muneerdar07@gmail.com*
*Author α : Senior Assistant Professor at University of Kashmir, received a BE in Electrical & Electronics Engineering from BITS-Pilani, India, MS in Computer Science from University of Oklahoma (USA) and PhD in Computer Science from the University of Kashmir, India.
E-mail : javedparvez1225@gmail.com*

It indicates that smart phone sales are continuously on rise and more and more people are becoming dependent on these devices. As these Smartphones are going to outnumber the world's total population in 2014, securing these devices has assumed paramount importance. Owners use their smart phones to perform tasks ranging from everyday communication with friends and family to the management of banking accounts and accessing sensitive Work related data. These factors, combined with limitations in administrative device control through owners and security critical applications like the banking transactions, make Android-based Smart phones a very attractive target for hackers, attackers and malware authors with almost any kind of motivation.

In this paper we analyze the security architecture of Smartphones in section II and identify the security loopholes of the current framework in section III. We then have a detailed description of the existing work done by the researchers in section IV. Finally we draw our conclusions in section V.

II. Security Framework of Smartphone

A Smartphone is an intricate combination of a mobile phone and a computing platform, with high-speed connectivity and powerful computing ability. Therefore, the Smartphone has necessary components of the computing platform: an operating system, applications and hardware. Furthermore, as a personal communication device, the Smartphone also often has multiple communication capabilities and ability to store large amounts of sensitive user data [2].

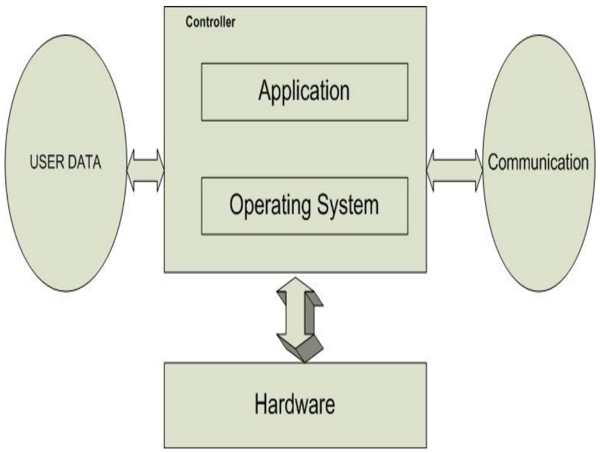

Figure 2 : Common Smartphone Architecture [2]

Android is a Linux-based platform programmed with Java and enhanced with its own security mechanisms tuned for a mobile environment. Android combines OS features like efficient shared memory, preemptive multi-tasking, Unix user identifiers (UIDs) and file permissions with the type-safe Java language and its familiar class library. The resulting security model is much more like a multi-user server than the sandbox found on the J2ME or Blackberry platforms. Unlike a desktop computer environment where a user's applications all run under the same UID, Android applications are individually siloed from each other.

Android applications run in separate processes under distinct UIDs each with distinct permissions. Programs can typically neither read nor write each other's data or code, and sharing data between applications must be done explicitly. The Android GUI environment has some novel security features that help support this isolation [10].

III. Limitations of Current Security Model

Android is based on open source framework and it comes with a pre-built suite of applications like dialer, address book, browser, etc. Developers can code their own applications and publish to the Android market after a self-signing phase that does not require any certifying authority i.e., developer can use self-created certificates to sign their applications. This helps in providing a wide range of applications and services to the device-holders. Since there is no support for root Certification Authorities in Android (Android is based on open source framework, while introducing root certification mechanisms contradicts the openness of Android) it is very difficult to scrutinize and/or block applications coming from unreliable sources. There is a higher possibility that an average Android user will easily be tricked by unsafe applications and will avoid the warning messages at time of installation.

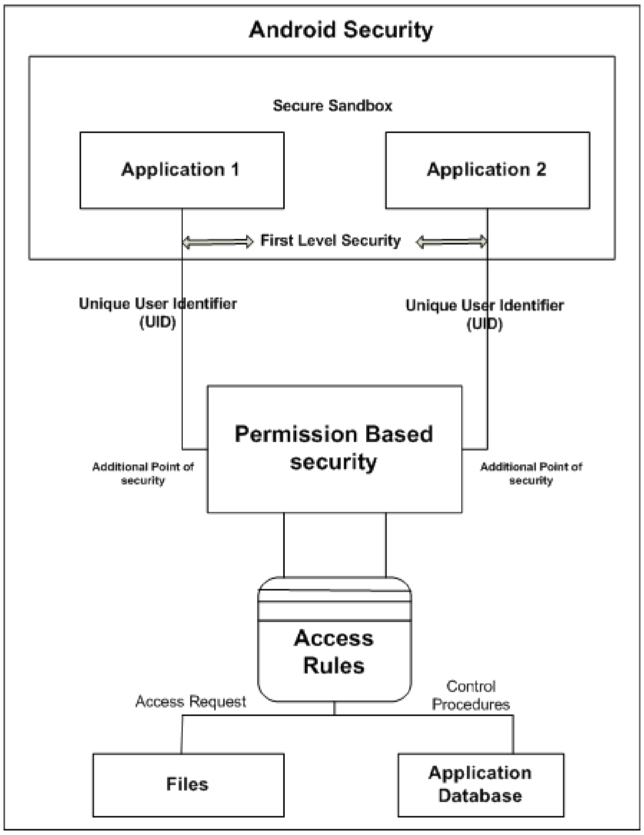

Figure 3 : Security Architecture of Android

The permission model is the core mechanism for securing access to various resources in Android. Although the permissions are categorized to different protection levels such as Normal, Dangerous, Signature and Signature-Or-System but the assignment of these protection levels is left to the developer's will and own understanding. This leads to a number of vulnerabilities in the permission model. When an application is installed on Android, the Android framework prompts the user with a list of required permissions, the user may grant all of the permissions in order to install the application or deny the permissions to decline the installation. Practically, there are a number of issues in such a model: 1) The user must grant all of the required permissions in order to install the application, 2) once the permissions are granted; there is no mechanism for further restricting an application to use the granted permissions, 3) there is no way of restricting access to the resources based on dynamic constraints as the permission model is based on install-time check only, 4) granted permissions can only be revoked by uninstalling the application. There are four security loopholes that endanger the information stored in a Smartphone [18]:

- The capability to sniff the data transmitted by any network where the device is connected.
- The lack of strong security control of user´s private information that permits malware to access the information stored in the device.

- Saved information is not stored encrypted within media.
- The lack of configurable firewalls integrated into the operating systems.

These are the most common security problems in the smart phones. Powerful hardware and advanced operating system with flexible APIs [23], [24] not only increase capability and functionality of smartphones but also present rising security threats to smartphones. Other features of smartphones also exacerbate threats to smartphones: higher bandwidth not only accelerates the Internet access, but also speeds up virus transmission; multiple peripheral interfaces not only increase Smartphone connectivity, but also provide much more avenues for virus injection. Compared with personal computers, smartphones also have always-on & always-connected mobility and are therefore hard to reinstall. Therefore, disabling or making smartphones nonfunctional as a result of attacks will have much greater impact and prove more costly than in the case of personal computers. A threat means potential violation of Smartphone security, which can be clustered into two categories according to its cause: vulnerabilities [5] and attacks [6]. Vulnerabilities mean weakness of smartphones and inabilities to withstand hostile environment effects. Attacks are any attempts to intentional destroy unauthorized use, maliciously modify or illegally obtain Smartphone assets. Generally, attackers bypass or exploit deficiencies of Smartphone security mechanisms to initiate negative activities. That is, **vulnerabilities** are the internal attributes of smartphones, while **attacks** are the outside offensive activities to smartphones. As a matter of fact, most attacks exploit vulnerabilities of smartphones.

a) Vulnerabilities

As comparatively complicated devices, smartphones inevitably have numerous vulnerabilities, which can lead to insecurity or be exploited by malicious persons to initiate attacks. Smartphone vulnerabilities typically include system defects, insufficient management of APIs, deficiency of user awareness and unsecure wireless channels, shown in Fig.4.

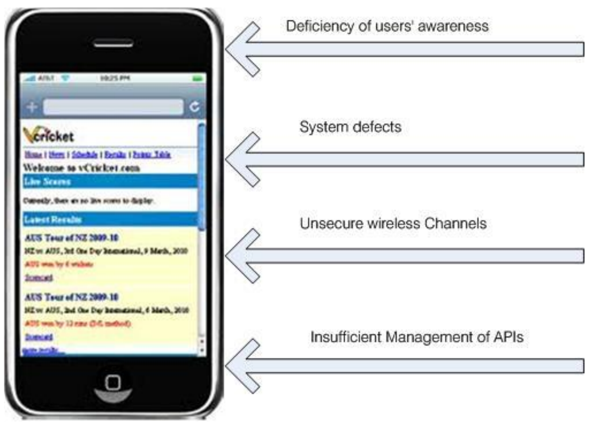
Deficiency of users' awareness
System defects
Unsecure wireless Channels
Insufficient Management of APIs

Figure 4 : Vulnerabilities of Smartphones

i. *Deficiency of User Awareness*

Some applications are installed without user confirmation or with limited information. Based on these situations, attackers can give deceptive information for applications infected by malicious codes. Users may install them without accurate or sufficient information. In addition, some sensitive operations, such as sending and receiving messages, deleting important files, activating wireless interfaces, can also be executed secretly. Consequently, Smartphone users cannot know about occurrences of malicious security-sensitive operations until the negative effects start appearing.

ii. *System Defects*

It is nearly impossible to detect and rectify all defects in Smartphone hardware and software. Some immediately non-conforming defects can be observed soon, but most other defects cannot be found for a certain time. Even if they are discovered, these defects, especially hardware defects are hard to be remedied. As a result, system defects can cause Smartphone abnormalities and malfunctions. Furthermore, malicious persons generally take advantage of existing system defects to initiate attacks and compromise Smartphone systems.

iii. *Unsecure Wireless Channels*

In wireless environments including cellular networks, user data and control signals transmitted between smartphones and network devices can be easily captured. If these data and signals are compromised, the transmitted information will be exposed.

iv. *Insufficient API Management*

The most distinct characteristic of smartphones is flexible APIs, which are used for application development [22] and installation. However, insufficient API management is also the main reason for malicious codes. Generally, Smartphone APIs are clustered into open APIs for third-party application developments and controlled APIs for remote maintenance. Controlled APIs have higher privileges, which can be used for remote system update, file erasure and information retrieval. If malicious persons obtain controlled APIs, they can initiate negative activities such as backdoor attacks. Even some open APIs may have inappropriate privileges so that they might be utilized to acquire certain privileges and initiate attacks.

b) Attacks

It is well known that smartphones have much valuable user data, especially **financial data** and **identification information**. Driven by economic benefits, many hackers have focused on smartphones and initiated multifarious attacks, shown in Fig.3:

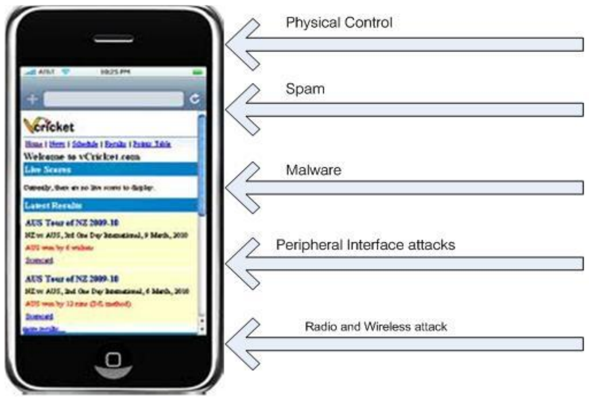

Figure 5 : Attacks to Smartphones

i. *Physical Control*

Due to portability and mobility, smartphones are likely to be lost or stolen. Then, sensitive information stored in smartphones including address book, communication records, usernames and passwords, etc. can be accessed directly. Incorrect disposals of old and damaged devices will cause similar problems.

ii. *Spam*

Spam is generally sent in SMS, MMS and email. VoIP and Instant Messaging (IM) have also become common ways for spamming. Spam may generally cause virus infection, economic loss, or worse influences.

iii. *Malware*

Malware [4], [5] is becoming the main threat to smartphones. Flexible APIs not only enrich application development, but also facilitate malware development. Meanwhile, powerful connectivity also aggravates malware spreading. In addition, smartphones can be infected by malicious codes during synchronization with personal computers or virus-infected storage media. Furthermore, malware can also be spread in a variety of ways, including Internet downloading, messaging services and Bluetooth communications. In reality, users are not always aware of downloaded applications' functions. *Even if applications have acquired explicit user consent, users may be unaware that the applications are executing malicious code.*

iv. *Backdoor*

Backdoor attacks mainly result from system bugs and disclosure of controlled APIs. Some operating systems have security loopholes such as insufficient authentication and inappropriate privileges. Based on these vulnerabilities, attackers can bypass security policies to access smartphones. In addition, if attackers have controlled APIs, they can also access smartphones like legitimate entities.

v. *Peripheral Interfaces Attacks*

Smartphones usually have many peripheral interfaces, such as Wi-Fi, Bluetooth, USB, etc. While peripheral interfaces can increase smartphones communication capabilities, unfortunately, they also become a popular steppingstone for outside attacks.

vi. *Radio and Wireless Attacks*

Due to the openness of wireless communications, attackers can easily initiate wireless attacks, which can be clustered into two categories: **active attacks** (spoofing, corruption, blocking, modifying, etc.) and **passive attacks** (sniffing, eavesdropping, etc.). Generally, passive attacks are used as a prelude to active attacks, by acquiring necessary information such as addresses and to identify vulnerabilities of potential targets.

IV. Related Research Work

a) Kirin

Enck, et al. [22] have proposed a framework known as Kirin – install-time certification mechanism – that allows the mobile device to enforce a list of pre-defined security requirements prior to installation process of an application. During installation of an application the Android framework informs the user regarding the resources that can be accessed by the application but it cannot reflect the possibility of using different combinations of permission in a malicious manner. The Kirin framework is contacted when installation process for an application package is initiated. Kirin utilizes the application's manifest file where all the required permissions are listed and uses the action string along with the permissions to construct a set of Prolog facts Although Kirin is one of the first security policy extension for Android platform, it suffers from the common limitation of Smartphone security systems i.e. the policy expressibility is not sufficient enough to express certain policies. Furthermore, the policies used by the Kirin framework are based on blacklisting and must be defined upfront. This means that certain set of permissions would be considered as dangerous by the policy writer but any combination of these permissions that is not explicitly termed as dangerous is treated safe by default.

b) SCanDroid

Fuchs, et al. [23] proposed SCanDroid framework for Android to perform information flow analysis on applications in order to understand the flow of information from one component to another component. Consider a case where an application request permission to access multiple data stores i.e., public data store and private data store. The application requires permission for reading the data from the private store and writing data to the public store. SCanDriod analyzes the information flow of the application and report whether the application will transfer the information in the private store to the public store or not. However, SCanDroid also suffers from the same limitation of security policy expressibility. In order to consider some information flow to be dangerous, the policy writers must define certain constraints prior to executing the policy. Similarly, if an information flow is not explicitly added to the set of constraints the framework will consider it to be safe.

c) Saint

Ongtang, et al. [13] proposed Saint – a framework that provides security policy constraints in a more expressive manner by defining install-time permission granting policies and runtime policies for inter-component communication. The framework places a number of dependency constraints on the permissions requested by the applications. These constraints may include the name of application, versions, signatures and set of other permissions. The effectiveness of Saint is based on its runtime policies for which reference monitors are used in the Android framework. The runtime policies are used to specify constraints for both the caller and callee applications. These constraints include permissions, configurations, signatures and/or the context in which the application is used e.g., time, location etc. The framework enables an application to protect and restrict its interface from being used by another application. However, this framework is not usercentric as it gives the option of policy specification to the application developers and not to the user. In our opinion, as the owner of the device, the decision to grant or deny access to the device resources should remain with the user and not with the application developers.

d) Apex

Apex [24] is an extension to the Android permission model that is more user-centric in allowing applications to access the device resources. Apex allows users to specify detailed runtime constraints to restrict the use of sensitive resources by applications. It is designed to overcome the limitation that the Android framework grants all the permissions to an application, which the application requests at install time. At install time the only way to deny the permissions requested by an application is to abort the installation. In the same way, the only way of revoking permissions once they are granted to an application is to uninstall the application. Contrary to this, Apex enables users to define conditions that must be fulfilled by an application in order to grant requested permissions to it. This means that it allows a subset of the requested permissions to be granted to the application at install-time. This way, user can start using the application with a limited number of permissions. The user may extend the granted permissions at a later stage. However, there are some limitations in the Apex framework. In the current Android architecture, the application developers assume that all the permissions that their application requests will be present in the manifest file. The developers often do not handle the unexpected security exceptions that are thrown when an application requests to access some resource(s) but the application does not have the required permissions to access it. If these exceptions are not properly handled – as may be the case in general – then we assume that most of the Android applications will not catch the exceptions and the exception will reach to the end of the call stack resulting in the termination of the thread.

e) Porscha

Ongtang, et al. [25] have proposed Porscha – a framework that enforces Digital Rights Management (DRM) policies – designed specifically for SMS, MMS and Email services allowing the content owners to restrict access to their content by specifying access control policies based on certain conditions like location and number of times to view a particular content etc. However, it is designed to facilitate different enterprises and government organizations with strictly controlled access policies.

f) CRePE

Conti, et al. [26] have proposed CRePE – a framework that enforces context-related fine-grained access control policies. It allows users to define policies that enable/ disable certain functionalities such as GPS, read SMS or Bluetooth discovery, based on the context of the device (e.g., location, noise, temperature, time, and nearby devices etc.). Furthermore, the context may also be defined by a trusted third party in scenarios where enterprise wide policy need to be deployed for all employees having Android smartphones. However, this framework only focuses on enabling/disabling of certain features of applications and cannot cope with the vulnerabilities that are formed by the permission usage across different applications.

g) XManDroid

Bugiel, et al. [27] have proposed XManDroid – extending monitoring on Android – to alleviate the problem of application-level privilege escalation attacks on Android. It analyzes the intercomponent communication mechanism among different applications in Android to ensure that these communication links comply with the predefined security policy. One of the major obstacles for the XManDroid is to define and maintain useful policies as well as policy exceptions. Some similar countermeasures against such malicious applications have already been proposed. Enck et al proposed "TaintDroid", a system-wide dynamic taint tracking system, in which multiple sources of sensitive data are tainted and the taint is used as a marker capable of real-time tracking of sensitive data [5]. They implemented "TaintDroid" and the evaluation results suggest that the overhead time for taint tracking is about 29% at most.

Takemori et al proposed the "white-list" measure which allows only secure and necessary applications to work [6]. In the white-list all approved applications are shown. In order to prevent malicious applications from intruding, any unlisted application will be immediately deleted even if it is installed. Kawabata et al pointed out the risk that attackers could execute Java method by using JavaScript downloaded from the server and it may in turn cause malicious behavior [7]. To counter this, they proposed to conduct a static analysis for Android applications with JavaScript to ascertain its threat level.

Chin et al mentioned vulnerability of interapplication Communication in Android [8]. They pointed out that some malicious components can eavesdrop and tamper with the "Intent" while sending and receiving a message between applications. They surveyed 100 applications and revealed that they undoubtedly have such vulnerabilities. To realize this security goal without adding unnecessary burden to developers, Harunobu Agematsu proposed to prepare a dedicated API called "ADMS API" and to create "Knowledge Database" which security manager could use to judge malicious behavior [4].

The United States National Security Agency has recently announced the commencement of the SEAndroid (Security Enhanced Android) project as an addition to the Android kernel [20]. Similar to the well known and widely deployed SELinux Linux kernel patch, the SEAndroid project aims to establish a fine grained Mandatory Access Control model. It is further adjusted and extended to meet the requirements which arise on the Android platform, e.g., to secure inter process communication [21]. Once integrated into Android, SEAndroid may indeed prevent some of the attacks presented. SEAndroid is still in a very early development stage. It is unknown when or if SEAndroid will be integrated into the default code base of the operating system.

V. Summary and Conclusion

We have elaborated the limitations in the current Android security model in detail and in previous section we have presented the existing research proposals for improving overall Android security model. In this section we detail some of the security requirements that need to be taken into consideration while designing security mechanisms for smartphones in general and Android in particular. In order to alleviate the limitations and further strengthen the Android security model, one of the most important security concerns for the current smartphones is the lack of a model that allow users to specify, at a fine-grained level, which of the phone's resources should be accessible to third party applications. To design policies that are fine-grained in expressibility and are targeted to cater application-specific requirements is one of the biggest challenges in proposing new security enhancements. It requires a pre-design analysis of real applications to gather a larger collection of likely scenarios where the fine-grained policies are applicable. While designing a new framework, it should be capable of specifying a set of detailed runtime constraints to restrict the use of sensitive resources by applications. For example, the user may want to restrict certain applications to access a particular resource in a particular context (e.g., time, location, and maximum number of usage etc.) without uninstalling the application. This could be achieved by designing a framework that allows granting selective permissions at install time as well as monitor the use of these permissions at runtime by employing certain usage control mechanisms. The growing number of malware vulnerabilities has augmented serious concerns over security models for the smartphones. The recent attacks discussed in [17], [19], [20] have shown how easily some of the Android security features can be overturned by the malware developers. While designing new framework or proposing enhancement to the Android security model, we should consider that the model should not be by-passable by the sophisticated malware and/or the applications installed on the device as well as new applications. The framework should be designed in such a manner that it can validate that the system is not tampered with. It should be able to prevent information leakage from the device in scenarios where a legitimate application is replaced with a similar one containing Trojans that spy on user's sensitive information such as location, or, logs the phone calls and transfers that information to a remote server.

References Références Referencias

1. Google bets on Android future. http://news.bbc.co.uk/2/hi/technology/7266201.stm
2. Hongwei Luo†‡, Guili He†, Xiaodong Lin§, and Xuemin(Sherman) Shen‡ "Towards Hierarchical Security Framework for Smartphones" First IEEE International Conference on Communications in China: Communications Theory and Security (CTS) 2012.
3. P. Ferrill, "Exploring the android api," *Pro Android Python with SL4A*, pp. 113–138, 2011.
4. Harunobu Agematsu, Junya Kani, Kohei Nasaka, Hideaki Kawabata "A proposal to realize the provision of secure Android applications" IEEE 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing.
5. J. Jamaluddin, N. Zotou and P. Coulton, "Mobile phone vulnerabilities: a new generation of malware," in *Consumer Electronics, 2004 IEEE International Symposium on*. IEEE, 2004, pp. 199–202.
6. William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth : "TaintDroid: An Information- Flow Tracking System for Realtime Privacy Monitoring on Smartphones", Proceedings of the 9th USENIX

Symposium on Operating Systems Design and implementation (OSDI'10), Canada, 2010.

7. Keisuke Takemori, Hideaki Kawabata, Takamasa Isohara, Ayumu Kubota, Jyunichi Ikeno: "Restriction framework for Android application -Whitelist-based installation", IPSJ (The Information Processing Society of Japan) SIG (The Special Interest Group) technical reports, 2011-CSEC-53-2, pp.1-6, 2011.5 (in Japanese).
8. Hideaki Kawabata, Takamasa Isohara, Keisuke Takemori, Ayumu Kubota: "Threat of Script abuse Android Permissions and Static Analysis", IPSJ SIG technical reports, 2011-CSEC-53-3, pp.1-6, 2011.5 (in Japanese).
9. A. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshopon Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 3–14.
10. Ruben Jonathan Garcia Vargas "Security Controls for Android" 2012 IEEE Fourth International Conference on Computational Aspects of Social Networks (CASoN) 215.
11. R. Rogers, J. Lombardo, Z. Mednieks, and B. Meike, *Android application development: Programming with the Google SDK*. O'Reilly Media, Inc., 2009.
12. C. Guo, H. Wang, and W. Zhu, "Smart-phone attacks and defenses," in *HotNets III*, 2004.
13. C. Dagon, T. Martin, and T. Starner, "Mobile Phones as Computing Devices: the Viruses Are Coming," IEEE Pervasive Computing, vol. 3, no. 4, 2004, pp. 11–15.
14. M. Goadrich and M. Rogers, "Smart smartphone development: ios versus android," in *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 2011, pp. 607–612.
15. SELinux project, "SEAndroid", August 2012.
16. S. Salah, S. Abdulhak, H. Sug, D. Kang, and H. Lee, "Performance analysis of intrusion detection systems for smartphone security enhancements," in Mobile IT Convergence (ICMIC), 2011 International Conference on. IEEE, 2011, pp. 15–19.
17. M. Pelino, Predictions 2010: Enterprise Mobility Accelerates Again, Forrester, 2009.
18. Angel Alonso Parrizas, SANS Institute, "Securely deploying Android devices", September 2011.
19. Jeter, L. Mani, M, Reinschmid, T., University of Colorado, "SmartPhone Malware: The danger and protective strategies", 2011 August.[20] Android Open Source Project, Google, "Android Security Overview", 2011.
20. S. Smalley, "SE Android release." SELinux Mailing List, Mailing List Archives (marc.info), January 2012. ttp://marc.info/?l=selinux&m=132588456202123&w=2.
21. National Security Agency, "SEAndroid Project Page,"January012.http://selinuxproject.org/page/SEAndroid.
22. W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proceedings of the 16th ACM conference on Computer and Communications Security. ACM, 2009, pp. 235–245.
23. A. Fuchs, A. Chaudhuri, and J. Foster, "Scandroid: Automated security certification of android applications," Manuscript, Univ. of Maryland, http://www.cs.umd.edu/˜avik/projects/scandroidascaa.
24. M. Nauman, S. Khan, and X. Zhang, "Apex: Extending android permission model and enforcement with user-defined runtime constraints," in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. ACM, 2010, pp. 328–332.
25. M. Ongtang, K. Butler, and P. McDaniel, "Porscha: Policy oriented secure content handling in android," in Proceedings of the 26th Annual Computer Security Applications Conference. ACM, 2010, pp. 221–230.
26. M. Conti, V. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," Information Security, pp. 331–345, 2011.
27. S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks," TR-2011-04, Technische Universit¨at Darmstadt, April. 2011., Tech. Rep.