Overview of Security on Mobile Devices

Lukáš Aron, Petr Hanáček
Faculty of Information Technology, Department of Intelligent Systems
Brno University of Technology

Abstract

This paper contains depth description of security models of modern mobile operating system like Android, iOS and Windows Phone. These security models are cornerstones of security on current platforms. Despite of different approaches of security they have a lot of in common. This paper also contains the most discussed security problem of nowadays, Malware. Descirption of malicious software is from Appliation-based view. However, modern operating system has strong protection against viruses and other types of infection through its security model, the weakes point of mobile devices are still users. These useres usually install additional software into their devices. This paper focuses on Android malware infection and provides a few protection methods against this type security threat.

Introduction

Mobile devices development has been enormous over past 20 years and its results are all around us. It has been a long time since mobile phones were used only for making a call or writing short text messages. Technologically advanced societies are trying to speed up and simplify any process that can be automated and to provide user an easy access to it. These processes may be implemented as applications on mobile devices and are aimed on helping people to finish daily tasks easily or more quickly. Such mobile devices could be smart phones, tablets, notebooks or similar devices which man can easily carry along with him. Recent years have witnessed an explosive growth in smartphone sales and adoption.

The software on these mobile devices consists of an operating system and applications that are installed on the device. The most widespread operating system is Android [1, 12] from Google, which will be the model example for the further explanation, mainly because of its popularity and open source properties, but the principles can be applied to every other platform being used. The paper is going to introduce and explain the principles of mobile threats appearing through all platforms of mobile operating systems. The introduction into security on mobile devices begins with security models of mobile platforms Android, iOS and Windows Phone, which are explained in the first part of this paper.

The next section of this paper is being aimed on the basic information about application-based mobile threats, and types of these threats in detail. Mobile threats are endangering the safety of individuals, companies, and if measures are not taken, than the cybercrime can have

DOI: 05. WSWAN.2015.1.4

an impact on the security of the whole society. First, we have to ask the question: Why do threats and attacks on mobile devices exist? The answer is simple since the motivation could be the same as for the attacks on desktop machines. Primary target of these attacks could be the secret information, whose gain could lead to stealing user's money, but attacker could be able to get an access to the computational power of the device, which should be also used for committing more cybercrime. The reason for emphasizing the security of mobile devices has its roots in this: while only experienced users were working with these devices 20 years ago, nowadays users that do not have any IT education and even small children are using modern technologies.

The simplest type of attack is to steal the device. The owner of the mobile device is generally the only user and that is the reason why there is not great emphasis on the physical security. This could be dangerous if the stolen device is the workstation of the user and the security threat to the whole company if the device is connected into corporate network.

Security Models

In this part of the paper are descriptions of security models or architectures of Android [6], iOS operating system[5] and Windows Phone [10]. Such security models or architectures are unique for each mobile platform, but they have a lot in common. For example all platforms enable creating applications by the third party developers. The differences are discussed in the following text.

Android Security Model

Android is an application execution platform for mobile devices comprised out of an operating system, core libraries, development framework and basic applications. Android operating system is built on top of a Linux kernel. The Linux kernel is responsible for executing core system services such as: memory access, process management, access to physical devices through drivers, network management and security. Atop the Linux kernel is the Dalvik virtual machine [9] or new one Art virtual machine along with basic system libraries. The Dalvik/Art VM is a register based execution engine used to run Android applications.

The Art virtual machine has been introduced in 2014 as successor of Dalvik. It is still in beta mode. Main differences between these implementation is that Dalvik is just-in-time compilation technique. The code is interpreted on demand as the application require. In contrast the Art virtual machine is working in ahead-of-time-compilation technique. That means, after downloading application the code is compiled into native code of the device. More information can be found in [9].

In order to access lower level system services, the Android provides an API through afore mentioned C/C++ system libraries. In addition to the basic system libraries, the development framework provides access the top level services, like content providers, location manager or telephony manager. This means that it is possible to develop applications which use the same system resources as the basic set of applications, like built-in web browser or mail client. However, such a rich development framework presents security issues since it is necessary to prevent applications from stealing private data, maliciously disrupting other applications or the operating system itself. In order to address the security issues, the Android platform implements a permission based security model.

The model is based on application isolation in a sandbox environment. This means that each application executes in its own environment and is unable to influence or modify execution of any other application. Application sandboxing is performed at the Linux kernel level. In order to achieve isolation, Android utilizes standard Linux access control mechanisms. Each Android application package (.apk) is on installation assigned a unique Linux user ID. This approach allows the Android to enforce standard Linux file access rights. Since each file is associated with its owner's user ID, applications cannot access files that belong to other applications without being granted appropriate permissions. Each file can be assigned read, write and execute access permission. Since the root user owns system files, applications are not able to act maliciously by accessing or modifying critical system components.

On the other hand, to achieve memory isolation, each application is running in its own process, i.e. each application has its own memory space assigned. Additional security is achieved by utilizing memory management unit (MMU), a hardware component used to translate between virtual and physical address spaces. This way an application cannot compromise system security by running native code in privileged mode, i.e. the application is unable to modify the memory segment assigned to the operating system.

The presented isolation model provides a secure environment for application execution. However, restrictions enforced by the model also reduce the overall application functionality. For example, useful functionalities could be achieved by accessing critical systems like: access to network services, camera or location services. Furthermore, exchange of data and functionalities between applications enhanced the capabilities of the development framework. The shared user ID and permissions are two mechanisms, introduced by the Android, which can be used to lift the restrictions enforced by the isolation model.

The mechanism must provide sufficient flexibility to the application developers but also preserve the overall system security. Two applications can share data and application components, i.e. activities, content providers, services and broadcast receivers. For example, an application could run and activity belonging to other application or access its files.

The shared user ID allows applications to share data and application components. In order to be assigned a shared user ID the two applications must be signed with the same digital certificate. In effect, the developers can bypass the isolation model restrictions by signing applications with the same private key. However, since there is not a central certification authority, the developers are responsible to keep their private keys secure. By sharing the user ID, applications gain the ability to run in the same process. The alternative to the shared user ID approach is to use the Android permissions. In addition to sharing data and components, the permissions are used to gain access to critical system modules. Each android application can request and define a set of permissions. This means that each application can expose a subset of its functionalities to other applications if they have been granted the corresponding permissions. In addition, each application can request a set of permissions to access other applications or system libraries. Permissions are granted by the operating system at installation and cannot be changed afterwards. There are four types of permissions: normal, dangerous, signature and signature-or-system. Normal permissions give access to isolated application-level functionalities. These functionalities have little impact on system or user security and are therefore granted without an explicit user's approval.

However, the user can review which permissions are requested prior to application installation. An example of a normal level permission is access to the phone's vibration hardware. Since it is an isolated functionality, i.e. user's privacy or other applications cannot be compromised, it is not considered a major security risk. On the other hand, dangerous permissions proved access to private data and critical systems. For example, by obtaining a dangerous permission, an application can use telephony services, network access, location information or gain other private user data. Since dangerous permissions present a high security risk, the user is promoted to confirm them before installation.

iOS Security Model

Unlike the Android security architecture, iOS security model [5] provides different philosophy for achieving mobile devices security and user's protection. The iOS application platform empowers developers to create new applications and to contribute to the application store. However, each application submitted by a third party developer is sent to the revision process. During the revision process the application code is analyzed by professional developers who make sure that the application is safe before it is released to the application store. However, such an application, when installed, gets all the permissions on a mobile device. Application might access local camera, 3G/4G, Wi-Fi or GPS module without user's knowledge. While Android lets each user handle its own security on their own responsibility, the iOS platform makes developers to write safe code using iOS secure API and prevents malicious applications from getting into the app store.

The iOS security APIs [4] are located in the Core Services layer of the operating system and are based on services in the Core OS – kernel layer of the operating system. Application that needs to execute a network task, may use secure networking functions through the CFNetwork API, which is also located in the Core Services layer.

The iOS security implementation includes a daemon called the Security Server that implements several security protocols, such as access to keychain items and root certificate trust management. The security Server has no public API. Instead, applications use the Keychain Services API and the Certificate, Key, and Trust services API, which in turn communicate with the Security Server. Keychain Services API is used to store passwords, keys, certificates, and other secret data. Its implementation therefore requires both cryptographic functions (to encrypt and decrypt secrets) and data storage functions (to store the secrets and related data in files). To achieve these aims, Keychain Services uses the Common Crypto dynamic library. CFNetwork is a high-level API that can be used by applications to create and maintain secure data streams and to add authentication information to a message. CFNetwork calls underlying security services to set up a secure connection. The Certificate, Key, and Trust Services API include functions to create, manage, and read certificates, add certificates to a keychain, create encryption keys, encrypt and decrypt data, sign data and verify signatures and manage trust policies.

To carry out all these services, the API calls the Common Crypto dynamic library and other Core OS-level services. Randomization Services provides cryptographically secure pseudorandom numbers. Such pseudorandom numbers are generated by a computer algorithm (and are therefore not truly random), but the algorithm is not discernible from the sequence. To generate these numbers, Randomization Services calls a random number generator in the Core OS layer. In case that the developers use the presented API properly and do not integrate malicious activities into the application, the application will be accepted into the App store.

Windows Phone Security Model

The Windows Phone security model [10] is the foundation for protecting the confidentiality, integrity, and availability of data and communications. This section provides details about the innovative security architecture of Windows Phone.

The Windows Phone security model is based on the principles of isolation and least privilege, and introduces the "chamber" concept. Each chamber provides a security boundary and, through configuration, an isolation boundary within which a process can run. Each chamber is defined and implemented using a policy system. The security policy of a specific chamber defines what operating system capabilities the processes in that chamber can access. There are four chamber types. Three of the chamber types have fixed permission sets, and the fourth chamber type is capabilities-driven. Apps that are designated to run in the fourth chamber type have capability requirements that are honored at installation and at run-time. The four chamber types are as follows:

- The Trusted Computing Base (TCB) chamber has the greatest privileges. It allows processes to have unrestricted access to most of the Windows Phone resources. The TCB chamber can modify policy and enforce the security model. The kernel and kernel-mode drivers run in the TCB chamber. Minimizing the amount of software that runs in the TCB is essential for minimizing the Windows Phone attack surface. Only Microsoft can add signed software components to the TCB chamber

- The Elevated Rights Chamber (ERC) can access all resources except security policy. The ERC is intended for services and user-mode drivers that provide functionality intended for use by other phone apps. The ERC is less privileged than the TCB chamber. Only Microsoft can add signed software components to the ERC chamber.

- The Standard Rights Chamber (SRC) is the default chamber for pre-installed apps. All apps that do not provide device-wide services run in the SRC. Microsoft Outlook Mobile 2010 is an example of an app that runs in the SRC.

- The Least Privileged Chamber (LPC) is the default chamber for all non-Microsoft apps that are available through the Windows Phone Marketplace. LPCs are configured using capabilities as described in the following section.

A capability is a resource for which user privacy, security, cost, or business concerns exist with regard to usage on Windows Phone. Examples of capabilities include geographical location information, camera, microphone, networking, and sensors. The LPC defines a minimal set of access rights by default. However, the LPC is dynamic and can be expanded using capabilities. Capabilities are granted during app installation, and their privileges cannot be elevated at run time. The capabilities–based least privilege model is advantageous for the reasons like attack surface reduction and user consent and control. Developers use the capability detection tool that is distributed with the Windows Phone Developer Tools to create the capability list for their app. The capability list is included in the app manifest in the app package (WMAppManifest.xml). Every app on Windows Phone runs in its own isolated chamber that is defined by the declared capabilities that the app needs to function. A basic set of permissions is granted to all apps, including access to isolated storage. There are no communication channels between apps on the phone other than through the cloud. Apps are isolated from each other and cannot access memory used or data stored by other applications, including the keyboard cache. In addition, Windows Phone does not allow apps to run in the background at any given time, which prevents hidden apps or spyware apps from preying on users. The moment a user switches to a different app on Windows Phone, the previously used app is put into a dormant state and its application state preserved. This approach ensures that an app cannot use critical resources or communicate with Internet–based services while the user is not using the app.Measures that help mitigate common risks associated with smartphones, such as exposure of confidential data to unauthorized users, build on the robust security architecture of Windows Phone. In addition, policy management that complements these measures is simplified by the integration of Windows Phone with existing Microsoft infrastructure.

Application-based Mobile Threats

The typical user today downloads or buys software and installs it without thinking much about its functionality. A few lines of description and some reviews might be enough to persuade the user to try it. Except for well-known software (written by software companies such as Microsoft, Google or Apple) or through the open-source community, it can be difficult to verify the authenticity of available software or vouch for its functionality. Shareware/trial-ware/free software is available for personal computers (PCs) and is now available for mobile devices, as well, and only requires one click to install it. Hundreds of software applications pop up every day in the marketplace from seasoned to newbie developers.

The problem is compounded for mobile devices, especially Android. With no rigorous security review (or gate) on multiple Android marketplaces, there are many opportunities for malicious software to be installed on a device. The only gate seems to be during the install process, when the user is asked to approve requested permissions. After that, the user's trust in an application is complete. Users, therefore, don't understand the full implications of the utilities and software that they install on their devices. Given the complexity and interdependencies of software installed, it can become confusing even for seasoned professionals to figure out if a software package is trustworthy. At these times, the need for reverse engineering becomes crucial.

Application-based threats or malicious applications are software codes designed to disrupt regular operations and collect sensitive and unauthorized information from a system or a user. Malware can include viruses, worms, Trojans, spyware, key loggers, adware, rootkits, and other malicious code[7]. The following behavior can typically be classified as malware:

- Disrupting regular operations: This type of software is typically designed to prevent systems from being used as desired. Behavior can include gobbling up all system resources (e.g., disk space, memory, CPU cycles), placing large amounts of traffic on the network to consume the bandwidth, and so forth.

- Collecting sensitive information without consent: This type of malicious code tries to steal valuable (sensitive) information – for example, key loggers. Such a key logger tracks the user's keys and provides them to the attacker. When the user inputs sensitive information (e.g. SSN, credit card numbers, and passwords), these can all potentially be logged and sent to an attacker.

- Performing operations on the system without the user's consent: This type of software performs operations on systems applications, which it is not intended to do – for example, a wallpaper application trying to read sensitive files from a banking application or modifying files so that other applications are impacted.

Identifying Android Malware

The content of this part is to identify behavior that can be classified as malware on Android devices. The question here is, how do we detect suspicious applications on Android and analyze them? There are a few steps of methodology identifying malware with source code of current application. There is a methodology called reverse engineering [8], but that methodology is not legal. If the user has source code of the application there are steps that the user should follow for identifying the malicious software on the mobile device:

- Source/Functionality: This is the first step in identifying a potentially suspicious application. If it is available on a non-standard source (e.g., a website instead of the official Android Market), it is prudent to analyze the functionality of the application. In many cases, it might be too late if the user already installed it on a mobile device. In any case, it is important to note the supposed functionality of an application, which can be analyzed throw next steps.

- Permissions: Now that user has analyzed and user understands the expected behavior of the application, it is time to review the permissions requested by the application. They should align with the permissions needed to perform expected operations. If an application is asking for more permissions that it should for providing functionality, it is a candidate for further evaluation.

- Data: Based on the permissions requested, it is possible to draw a matrix of data elements that it can have access to. Does it align with the expected behavior? Would the application have access to data not needed for its operations?
- Connectivity: The final step is analyzing the application code itself. The reviewer needs to determine if the application is opening sockets (and to which servers), ascertain what type of data is being transmitted (and if secured), and see if it is using any advertising libraries, and so forth.

The attackers usually do not have access to original source code without reverse engineering. The easiest way for modifying application is to get the source code and add some malicious behavior. This approach is widely used, but there is another type of adding malicious behavior, which can be done without access to source code of the mobile application. This technique is not generally used by a typical user or developer. A person using the techniques covered here is probably attempting one of the following (which is unethical if not illegal):

- To add malicious behavior: It should be noted that doing this is illegal. Malicious users can potentially download an Android application (apk), decompile it, add malicious behavior to it, repackage the application, and put it back on the Web on secondary Android markets. Since Android applications are available from multiple markets, some users might be lured to install these modified malicious applications and thus be victimized.
- To eliminate malicious behavior: The techniques listed here can be used to analyze suspicious applications, and, if illegal/malicious behavior is detected, to modify them and remove the illegal/malicious behavior. Analyzing an application for malicious behavior is fine and necessary for security and forensics purposes. However, if there is indeed such behavior detected, users should just remove the application and do a clean install from a reliable source.
- To bypass intended functionality: A third potential use of the techniques listed here could be to bypass the intended functionality of an application. Many applications require a registration code or serial key before being used or they can only be used for a specified trial period or show ads when being used. A user of these techniques could edit small code and bypass these mechanisms.

Recommended Security Practices for Mobile Devices

In the previous section were reviewed common threats to mobile devices and some of the mitigation steps one can take. In this section is covered in detail how to configure (harden) an Android device to mitigate the risks. These recommendation come from [2, 11, 3]. Security practices for mobile devices can be divided into four main categories:

- Policies and restrictions on functionality: Restrict the user and applications from accessing various hardware features (e.g., camera, GPS), push configurations for wireless, Virtual Private Network (VPN), send logs/violations to remote server, provide a whitelist of applications that can be used, and prevent rooted devices from accessing enterprise resources and networks.

- Protecting data: This includes encrypting local and external storage, enabling VPN communications to access protected resources, and using strong cryptography for communications. This also should include a remote wipe functionality in the case of a lost or stolen device.
- Access controls: This includes authentication for using the device (e.g., PIN, SIM password) and per-application passwords. A PIN/Passcode should be required after the device has been idle for few minutes (the recommendation is 2–5 minutes).
- Applications: This includes application-specific controls, including approved sources/markets from which applications can be installed, updates to applications, allowing only trusted applications (digitally signed from trusted sources) to be installed, and preventing services to backup/restore from public cloud-based applications.

Out of the box, Android does not come with all desired configuration settings (from a security viewpoint). This is especially true for an enterprise environment. Android security settings have improved with each major release and are fairly easy to configure. Desired configuration changes can be applied either locally or can be pushed to devices by Exchange ActiveSync mail policies. Depending on the device manufacturer, a device might have additional (manufacturer or third-party) tools to enhance security.

Unauthorized device access

As mentioned earlier in the paper, lack of physical control of mobile devices is one of the main concerns for a user and for an enterprise. The risk arising out of this can be mitigated to a certain extent through the following configuration changes:

Setting up a screen lock and SIM lock

After enabling this setting, a user is required to enter either a PIN or a password to access a device. There is an option to use patterns, although it is not recommend it. Recommendation for a strong password is an 8-digit PIN. Once "Screen Lock" is enabled, the automatic timeout value should be updated as well. Turning on the "SIM card lock" makes it mandatory to enter this code to access "phone" functionality. Without this code, one would not be able to make calls or send SMS messages.

Remote wipe

System administrators can enable the "Remote Wipe" function through Exchange ActiveSync mail policies. If a user is connected to the corporate Exchange server, it is critical to enable this feature in case the device is lost or stolen. There are other settings that can be pushed as well (e.g., password complexity). These are covered later in this paper. Remote Wipe essentially wipes out all data from the phone and restores it to factory state. This includes all e-mail data, application settings, and so forth. However, it does not delete information on external SD storage.

Conclusion

In this paper was described overall about mobile security threats and possible vulnerabilities. There are modern operating systems with strong security background which are provided to the users. There is nothing more important than the safety of the user's data. In these days there are a lot of known vulnerabilities in these operating systems, applications, internet browsers and specific teams and developers working on issues trying to fix known problems. However, there is the weakest point at this security and that point is always the user of the current device. There is not necessary that the attacker is a developer or technical educated person, it could be anyone who knows something personal and can deceive the user. For discussed platforms exist the additional adjustments which break the basic security model. This is usual called the "rooting" the device. It is because the operation systems are based on Linux or Unix kernel and the administrator or superpower user is called root. Another name for the same unlocking device could be jail-break (mainly for iOS platform). This adjustment can bring some more power to the user for settings or installing application from the other sources than is usual, but there are always the risk. The risk is always related to the security of the current mobile device.

Acknowledgement

This article was created within the research plan of the Security-Oriented Research in Information (MSM0021630528). This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by the project Advanced Secured, Reliable and Adaptive IT (FIT-S-11-1).

References

[1] Ed Burnette. *Hello, Android: introducing Google's mobile development platform.* Pragmatic Bookshelf, 2009.

[2] Jesse Burns. Developing secure mobile applications for android, 2008.

[3] Guiran Chang, Chunguang Tan, Guanhua Li, and Chuan Zhu. Developing mobile applications on the android platform. In *Mobile multimedia processing*, pages 264–286. Springer, 2010.

[4] Cedric Halbronn and Jean Sigwald. iphone security model & vulnerabilities. In *Proceedings of Hack in the box sec-conference. Kuala Lumpur, Malaysia*, 2010.

[5] Andrew Hoog and Katie Strzempka. *iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices.* Elsevier, 2011.

[6] DING Li-ping. Analysis the security of android. *Netinfo Security*, 3:011, 2012.

[7] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin. Attacks on webview in the android system. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 343–352. ACM, 2011.

[8] Ralf Mitsching, Carsten Weise, Stefan Kowalewski, Alexander Michailidis, Uwe Spieth, Bernd Hedenetz, Dominik Franke, Stefan Kowalewski, Carsten Weise, Daniel Merschen, et al. Inferring definite counterexamples through. In *NASA Formal Methods Symposium (NFM 2012)*, volume 1, pages 435–440. Springer, 2012.

[9] Hyeong-Seok Oh, Beom-Jun Kim, Hyung-Kyu Choi, and Soo-Mook Moon. Evaluation of android dalvik virtual machine. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems*, pages 115–124. ACM, 2012.

[10] Thomas Schaefer, Hans Höfken, and Marko Schuba. Windows phone 7 from a digital forensics' perspective. In *Digital Forensics and Cyber Crime*, pages 62–76. Springer, 2012.

[11] Jeff Six. *Application Security for the Android Platform: Processes, Permissions, and Other Safeguards.* " O'Reilly Media, Inc.", 2011.

[12] Welderufael Berhane Tesfay, Todd Booth, and Karl Andersson. Reputation based security model for android applications. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 896–901. IEEE, 2012.