Security Threats on Mobile Devices

Lukas Aron
Brno University of Technology, Czech Republic

ABSTRACT
This chapter contains basic introduction into security models of modern operating system like Android and iOS. There are described the methods of attacks to the mobile devices. Such attacks consist of application based threats and vulnerabilities, network based attacks and internet browser vulnerabilities. The following section contains description of defensive strategies and steps for securing the device. There is also section about securing mobile device for enterprise environment. At the end of this chapter are discussed recommendation for security practices for mobile devices.

Keywords: Mobile threats, cybercrime, security, vulnerabilities, Android, iOS

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 (Burnette; Tesfay, Booth, and Andersson) 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. There are also covered topics like securing Android for enterprise environment or recommendation security practices for mobile devices. The introduction into security on mobile devices begins with security models of mobile platforms Android and iOS, which are explained in the first part of this chapter.

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 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 could 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.

All security rules which were previously applied to personal or business computers or other non-mobile devices are now being applied to mobile devices. These rules are usually stricter, because owner of the device is also the main user (usually the only user) and may transmit sensitive information outside a secure area (e.g. home or office). It is necessary to refuse an access to this device by unauthorized users.

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. These problems related to networks and enterprise environment are covered after the section about application-based mobile threats.

There are a lot of types of security threats on mobile devices, but the weakest point is always non-expert user. In this chapter are discussed these threats that the user cannot control or can be deceived by the attacker. There are also cover defensive strategies and steps what to do for protecting the mobile devices as much as it possible. There are also recommendation security practices for mobile devices in the almost last part of this chapter.

The first section of this chapter is introduction into security models. Such models of modern mobile operating systems and compare them. The second part is targeted to application-based mobile threats and there are covered the most discussed security threats like malware, spyware, and privacy leak threats. After section about application-based mobile threats follow the section with caption defensive strategies. These strategies are steps for protection the mobile device. Different types of mobile threats which come from classical computers are browsers threats. These types of threats are well known from desktops and mobile devices are usually connected to the internet through internet browser which has almost the same vulnerabilities like the desktop one.

Next section is targeted to enterprise environment and how to secure this segment and follows the recommendations and security practices for mobile devices. The last part deals with conclusion about the mobile security in these days and to the near future.

SECURITY MODELS

In this part of the chapter are descriptions of security models or architectures of Android (Li-ping) and iOS operating system. Such security models or architectures are unique for each mobile platform, but they have a lot in common. For example both 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 (Oh et. all) 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 (Oh et. all).

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.

Signature permission can be granted to the application signed with the same certificate as application declaring the permission. The signature permission is in effect a refinement of the shared user ID approach and provides more control in sharing application data and components. On the other hand, signature or system permission extends the signature permission by granting to the applications installed in the Android system image. However, caution is required since both the signature and signature or system permissions will grant access rights without asking for the user's explicit approval. The source of this section is (Enck et. all).

iOS Security Model

Unlike the Android security architecture, iOS security model (Hoog and Strzempka) 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 (Halbronn and Sigwald) 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 deamon 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.

Summary

This part of the chapter was introduction into modern operating systems and their security models. There were description two of the most popular mobile operating systems – Android and iOS. They are using different approach for securing user's data. Android promoted the power and the responsibility of security for applications to developers. Developer of the application should use the recommendations which are written on the official website of the Android operating system. This approach has strong and also weak sites in comparing to iOS. The weakest point is that, the developers have the absolute power for the application security and user with no experience has very small chance to defend or protect the device. For example user wants some application and do not care about permissions what he confirmed, because without approval these permissions, application will not be installed. In contrast the strongest point is the same approach, but from the another point of view. The user has the power to deny installation the application, because he deny to using current application with these permissions. For example user does not want to allow the access to the contact list or messages for specific application and has the power to stop the installation process.

Another concept has the iOS, each application submitted by a third party developer is sent to the revision process. That is strong protection before malicious application, but the user has no power to deny specific permission.

Both platforms has own ways for protecting user's. Despite the protection they are provided, there are still possible security threats and malicious software which is the main problem of the present security on mobile devices. In the next part are described malicious applications and application-based threats.

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 (Li, Gu and Luo). The following behavior can typically be classified as malware:

1. 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.
2. 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.
3. 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 (Franke et. all), 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:

1. 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 steps 2 to 4.
2. 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.
3. 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?
4. 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):

1. **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.
2. **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.
3. **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.

DEFENSIVE STRATEGIES

In this section are covered five main strategies to prevent reverse engineering of an application or to minimize information leakage during the reverse engineering process. Defensive strategies are derived from (Misra)

Perform code obfuscation

Code obfuscation is the deliberate act of making source code or machine code difficult to read or understand by humans and thus making it a bit more difficult to debug and reverse engineer only from executable files. Companies use this technique to make it harder for someone to steal their IP or to prevent tampering.

Most Android applications are written in Java. Since Java code gets compiled into byte code (running on a VM) in a class file, it is comparatively easier to reverse engineer it or to decompile it than binary executable files from C/C++. Consequently, we cannot rely only on code obfuscation for protecting intellectual property or user's privacy. We need to assume that it is possible for someone to decompile the apk package and more or less get access to the source code. Instead of relying completely on code obfuscation, we suggest relying on "Server Side Processing", where possible (covered in the following section).

One of the freely available Java obfuscators that can be used with Android is ProGuard (Hoog). ProGuard shrinks and obfuscates Java class files. It is capable of detecting and removing unused classes, fields, methods, and so forth. It can also rename these variables to shorter (and perhaps meaningless) names. Thus, the resulting apk package files will require more time to decipher. ProGuard has been integrated into the Android-built system. It runs only when an application is built in the release mode (and not in the debug mode). ProGuard might not be one of the best obfuscators out there for Java. However, it is something that one should definitely use in the absence of other options.

Perform server side processing

Depending on the type of application, it might be possible to perform sensitive operations and data processing on the server side. For example, for an application that pulls data from the server to load

locally (e.g., Twitter, Facebook), much of the application logic is performed on the server end. Once the application authenticates successfully and the validity of the user is verified, the application can rely on the server side for much of the processing. Thus, even if compiled binary is reverse engineered, much of the logic would be out of reach, as it will be on server side.

Perform iterative hashing and use salt

Hash functions can be susceptible to collision. In addition, it might be possible to brute force hash for weaker hash functions. Hash functions make it very difficult to brute force (unless user is a government agency with enormous computing power) while providing reasonably high collision resistance.
The SHA-2 family fits this category. A stronger hash can be obtained by using salt.

In cryptography, a salt consists of random bits and is usually one of the inputs to the hash function (which is one way and thus collision resistant). The other input is the secret (PIN, passcode, or password). This makes brute force attacks more difficult, as more time/space is needed. The same is true for rainbow tables. Rainbow tables are a set of tables that provide precomputed password hashes, thus making it easier to obtain plaintext passwords. They are an example of space-time or time-memory trade off (i.e., increasing memory reduces computation time).

In addition, it is recommended to use iterative hashing for sensitive data. This means simply taking the hash of data and hashing it again and so on. If this is done a sufficient number of times, the resultant hash can be fairly strong against brute force attacks in case an attacker can guess or capture the hash value.

Choose the right location for sensitive information

The location of sensitive information (and access to it) matters as much as the techniques described above. If we store strong hashes at a publicly accessible location (e.g., values.xml or on an SD card or local file system with public read attributes to it), then we make it a bit easier for an attacker. Android provides a great way to restrict access - data can only be explicitly made available through permissions wherein, by default, only the UID of the app itself can access it.

An ideal place for storing sensitive information would be in the database or in preferences, where other applications don't have access to it.

Cryptography

In the iterative hashing section were discussed how to make a user's passwords or sensitive information stronger through the use of cryptography (hashing and salt). Cryptography can also be used to protect a user's data. There are two main ways of doing this for Android:

1. Every application can store data in an encrypted manner (e.g., the user's contact information can be encrypted and then stored in a sqlite3 database).
1. Use disk encryption, wherein everything written to the disk is encrypted or decrypted on the fly.

System administrators prefer full-disk encryption, so as not to rely on developers to implement encryption capabilities in their Apps

Summary

Access Control (relying on the OS to prevent access to critical files), cryptography (relying on encryption as well as hashing to protect confidential data [e.g., tokens] and to verify the integrity of an application), and code obfuscation (making it difficult to decipher class files) are the main strategies that one should

leverage to prevent the reverse engineering of applications. In this part of the chapter, were discussed best practices to prevent reverse engineering as well as the potential leaking of sensitive information through it.

BROWSER SECURITY

Mobile devices has access to the internet network and this is usually provides by internet browser which is well known from classical PC. Such browser has only few differences from the PC, for example less consumption of energy or power of the device and also worse performance. This part describes HTML and browser security on mobile devices. There are covered different types of attacks possible, as well as browser vulnerabilities.

Mobile HTML security

The increasing adoption of mobile devices and their use as a means to access information on the Web has led to the evolution of websites. Initially, mobile browsers had to access information through traditional (desktop-focused) websites. Today most of these websites also support Wireless Application Protocol (WAP) technology or have an equivalent mobile HTML (trimmed-down sites for mobile devices).

WAP specification defines a protocol suite that enables the viewing of information on mobile devices. The WAP protocol suite is composed of the following layers: Wireless Datagram Protocol (WDP), Wireless Transport Layer Security (WTLS), Wireless Transaction Protocol (WTP), Wireless Session Protocol (WSP), and Wireless Application Environment (WAE). The protocol suite operates over any wireless network.

In a typical Internet or WWW model, there is a client that makes a request to a server. The server processes the request and sends a response (or content) back to the client. This is more or less same in the WAP model, as well. However, there is a gateway or proxy that sits between the client and the server that adapts the requests and responses (encodes or decodes) for mobile devices. WAP 2.0 (Tull) provides support for richer content and end-end security than WAP 1.0.

WAP 1.0 did not provide end-end support for SSL/TLS. In WAP 1.0, communications between a mobile device and WAP gateway could be encrypted using WTLS. However, these communications would terminate at the proxy/gateway server. Communications between the gateway and application/HTTP server would use TLS/SSL. This exposed WAP 1.0 communications to MITM (Man-In-The-Middle) attacks. In addition, all kinds of sensitive information would be available on the WAP gateway (in plaintext). This meant that a compromise of the WAP gateway/proxy could result in a severe security breach. WAP 2.0 remediates this issue by providing end-end support for SSL/TLS.

WAP and Mobile HTML sites are also susceptible to typical Web application attacks, including Cross-Site Scripting, SQL Injection, Cross-Site Request Forgery, and Phishing. Mobile browsers are fully functional browsers with functionality that rivals desktop versions. They include support for cookies, scripts, flash, and so forth. This means that users of mobile devices are exposed to attacks similar to those on desktop/laptop computers. A good source for detailed information on these attacks is the Open Web Application Security Project (OWASP.

Cross-Site scripting

Cross-Site Scripting (XSS) (Backes, Gerling and Styp-Rekowsky) allows the injection of client-side script into web pages and can be used by attackers to bypass access controls. XSS attacks can result in attackers obtaining the user’s session information (such as cookies). They can then use this information to bypass access controls. At the heart of XSS attacks is the fact that untrusted user input is not thoroughly

vetted and is used without sanitization/escaping. In the case of XSS, user input is not sanitized for and is then either displayed back to the browser (reflected XSS) or stored (persistent XSS) and viewed later.

Mobile sites are as prone to XSS attacks as their regular counterparts, as mobile HTML sites might have even less controls around validating/sanitizing user input. Treating mobile HTML sites like regular websites and performing proper validation of user input can prevent a site from being vulnerable to XSS attacks.

SQL Injection

SQL injection (Clarke; Johari and Sharma; Bavani) allows the injection of an SQL query from a client into an application. A successful SQL query (or attack) can provide attackers with sensitive information and enable them to bypass access controls, run administrative commands, and query/update/delete databases.

At the heart of SQL injection attacks is the fact that untrusted user input is directly used in crafting SQL queries without validation. These SQL queries are then executed against the backend database.
Similar to XSS, mobile HTML and WAP sites are prone to SQL injection attacks. Mobile sites might have the same flaws as their desktop counterparts, or, even worse, they might not be performing the validation of user input when accepting inputs through mobile sites. Using parameterized queries or stored procedures can prevent SQL injection attacks.

Cross-site request forgery

A Cross-Site Request Forgery (CSRF, XSRF) (Bhavani) attack results in unwanted (unauthorized) commands from a user already authenticated to a website. The website trusts an authenticated user and, therefore, commands coming from him, as well. In CSRF, the website is the victim of the trust in the user, whereas in XSS, the user is the victim of the trust in the server/website.

It is typical for a user to be authenticated to multiple websites on a mobile device. Thus, CSRF attacks are possible, just as they are on desktop or laptop computers. In addition, small interface and UI layouts can disguise CSRF attacks (e.g., an e-mail with a URL link) and trick the user into performing unwanted operations on a website.

Phishing

Phishing attacks target unsuspecting users and trick them into providing sensitive information (e.g., SSN, passwords, credit card numbers, etc.). Through social engineering, attackers trick users to go to legitimate-looking websites and perform certain activities. Users trusting the source for this request (e.g., typically in an e-mail) performs the recommended operation and, in turn, provides an attacker with sensitive data.

As an example, a user gets an e-mail that seems legitimate and looks like it came from his bank. It is requesting the user to change his password due to a recent security breach at the bank. For his convenience, the user is provided with a URL to change his password. On clicking the link, the user is taken a website that looks like the bank's website. The user performs the password-reset operation and, in turn, provides the current password to the attacker.
Such attacks are even more difficult for users to recognize on mobile devices. Due to small UI real estate, users cannot really read the entire URL that they are viewing. If they are being redirected to a website, they would not be able to tell it easily on a mobile device.

Differences between legitimate and fake websites are not easy to distinguish on a small UI screen of mobile devices. If URLs are disguised (e.g., tiny URL) or if these are URLs that are sent through a Short Message Service (SMS) message (tiny URL via SMS), it is even more difficult to distinguish between legitimate and fake requests. Many users (even ones who are aware of such attacks) can be tricked into going through with an attack. As mentioned in the previous chapter, Quick Response (QR) codes can also be used for such attacks.

Browser vulnerabilities

As of the writing of this chapter, there are ~200+ Common Vulnerabilities and Exposures (CVEs) related to the Android platform (search cve.mitre.org for "android"). Of these, many are related to browsers (either built-in browsers or downloadable browsers, such as Firefox). There are a few examples of browser related vulnerabilities which are connected to Android or iOS. The following information are derived from (Luo et. all; Enck et all)

1. CVE 2008-7298: The Android browser in Android cannot properly restrict modifications to cookies established in HTTPS sessions, which allows man-in-the-middle attackers to overwrite or delete arbitrary cookies via a Set-Cookie header in an HTTP response. This is due to the lack of the HTTP Strict Transport Security (HSTS) enforcement
2. CVE 2010-1807: WebKit in Apple Safari 4.x before 4.1.2 and 5.x before 5.0.2; Android before 2.2; and webkitgtk before 1.2.6. Does not properly validate floating-point data, which allows remote attackers to execute arbitrary code or cause a denial of service (application crash) via a crafted HTML document, related to nonstandard NaN representation
3. CVE 2010-4804: The Android browser in Android before 2.3.4 allows remote attackers to obtain SD card contents via crafted content:// URIs, related to (1) BrowserActivity.java and (2) BrowserSettings.java in com/android/browser
4. CVE 2011-2357: Cross-application scripting vulnerability in the Browser URL loading functionality in Android 2.3.4 and 3.1 allows local applications to bypass the sandbox and execute arbitrary Javascript in arbitrary domains by (1) causing the MAX_TAB number of tabs to be opened, then loading a URI to the targeted domain into the current tab, or (2) making two startActivity function calls beginning with the targeted domain's URI followed by the malicious Javascript while the UI focus is still associated with the targeted domain
5. CVE 2012-3979: Mozilla Firefox before 15.0 on Android does not properly implement unspecified callers of the android_log_print function, which allows remote attackers to execute arbitrary code via a crafted web page that calls the JavaScript dump function

Drive-by Downloads

Drive-by downloads have been an issue with computers for some time. However, they are starting to migrate to mobile devices. A drive-by download is basically malware that gets downloaded and often installed when a user visits an infected website.

Recently, we saw the first drive-by download malware for Android (named "NonCompatible"). When visiting an infected website, the browser could download this malware file. However, it cannot install itself without user intervention. In addition, installation from non-trusted sources needs to be enabled for the user to install this malware. An attacker can disguise such a download by renaming it as a popular Android application or updates to Android itself.

Users are willing to install such files without much thought and, thus, end up infecting their devices with malware. As long as "side loading" and installation of applications from "non-trusted" sources is disabled, such malware should not be able to impact Android devices.

SECURING ANDROID FOR THE ENTERPRISE ENVIRONMENT

This part of the chapter contains security concerns for deploying Android and Android applications in an enterprise environment (Wei et. all). At first there is review of security considerations for mobile devices, in general, as well as Android devices, in particular. Then move on to cover monitoring and compliance/audit considerations. This part is derived from (Kodeswaran et. all).

Android in Enterprise

From an enterprise perspective, there are different ways of looking at Android in the environment, with the main being the following three: deploying Android devices, developing Android applications, and the implications of allowing Android applications in the environment.
The deployment of Android devices and applications is primarily an IT function, whereas developing secure Android applications is part of either development/engineering teams or IT-development teams.

Security concerns for Android in Enterprise

Today's mobile devices, including Android mobile devices like phones and tablets, are evolving at a rapid rate in terms of hardware and software features. Our assessment of threats, as well as security controls, has not kept up with the evolution of these features. These devices need more protection due to the features available on them, as well as the proliferation of threats to them. Before such devices can be deployed in an enterprise (or applications developed), it is essential that is carefully considered threats to mobile devices, as well as to enterprise resources arising from mobile devices (and users). This can be done using a threat model. In threat modeling, we analyze assets to protect, threats to these assets, and resulting vulnerabilities.

Lack of physical control of device

Mobile devices are physically under the control of end users (not system administrators or security professionals). The fact that a device is with the user pretty much all the time increases the risk of compromise to an enterprise's resources. From shoulder surfing to the actual loss of the physical device, threats arise from the lack of physical control of these devices. Mobile devices are more likely to be lost, stolen, or are temporarily not within the user's immediate reach or view. Enterprise security should assume that once stolen or lost, these devices could fall into malicious hands, and thus security controls to prevent disclosure of sensitive data must be designed with this assumption.

Considering the worst-case scenario in which a lost or a stolen device falls into malicious hands, the best way to prevent further damage will be to encrypt the mobile device (if the storing of sensitive data is allowed) or not allowing devices to access sensitive information (not really possible with Android smartphones). To prevent shoulder surfing, it might be prudent to use privacy screens (yes, there are ones for phones). In addition, a screen lock (requiring a password/PIN) should be a requirement for using these devices, if access to enterprise resources is desired. The best practice would be to authenticate to a different application each time one uses it, although this is tedious, and, most likely, users will not adhere to this (imagine logging into the Facebook application on an Android device every time one uses it).

Use of "user-owned" untrusted devices

Many enterprises are following a BYOD (bring your own device) model. This essentially means that users will bring their own mobile device (which they purchase) and use it to access company resources. This poses a risk because these devices are untrusted (and not approved) by enterprise security, and one

has to rely on end users for due diligence. Thus, the assumption that all devices are essentially untrusted is not far-fetched.

Security policies need to be enforced even if these devices are owned by the users. In addition, these devices and applications on them need to be monitored. Other solutions include providing enterprise devices (which have a hardened OS and preapproved applications and security policies) or allowing user-owned devices, with sensitive resources being accessed through well-protected sandboxed applications.

Connecting to "unapproved and untrusted networks"

Mobile devices have multiple ways to connect: cellular connectivity, wireless, Bluetooth connections, Near Field Communication (NFC), and so forth. An enterprise should assume that any or all of these means of connectivity are going to be employed by the end user. These connectivity options enable many types of attacks: sniffing, man-in-the-middle, eavesdropping, and so forth.

An example of such an attack would be the end user connecting to any available (and open) Wi-Fi network and thus allowing an attacker to eavesdrop on communications (if not protected). Making sure communications are authenticated before proceeding and then encrypted can effectively mitigate risk from this threat.

Use of untrusted applications

This essentially replicates the problem on desktop/laptop computers. End users are free to install any application they choose to download. Even if the device is owned and approved by an enterprise, users are likely to install their own applications (unless prevented by the security policy for the device). For Android, a user can download applications from dozens of application markets or just download an application off the Internet.

There are several options for mitigating this threat. An enterprise can either prohibit use of third-party applications through security policy enforcement or through acceptable use policy guidelines. It can create a whitelist of applications that users are allowed to install and use if they would like to access company resources through their Android devices. Although this might prevent them from installing an application (e.g., Facebook), they might still be able to use this application through other means (e.g., browser interface). The most effective mitigating step here is educating the end user, along with policy enforcement. The monitoring of devices is another step that can be taken.

Connections with "untrusted" systems

Mobile devices synchronize data to/from multiple devices and sources. They can be used to sync e-mails, calendars, pictures, music, movies, and so forth. Sources/destinations can be the enterprise's desktops/laptops, personal desktops/laptops, websites, and increasingly, these days, cloud-based services. Thus, one can assume any data on the device might be at risk.

If the device is owned by the enterprise, security policies on the device itself can be enforced to prevent it from backing up or synchronizing to unauthorized sources. If the user owns the device, awareness and monitoring (and maybe sandbox applications) are the way to go.

Unknown content

There can be a lot of untrusted content on mobile devices (e.g., attachments, downloads, Quick Response (QR) codes, etc.). Many of these will be from questionable or unknown sources and can pose risks to user and enterprise data. Take, for example, QR codes. There can be malicious URLs or downloads hidden throughout these codes, but the user might not be aware of these, thus falling victim to an attack.

Installing security software (anti-virus) might mitigate some risk. Disabling the camera is another option to prevent attacks such as those on QR codes. Awareness, however, is the most effective solution here.

Use of GPS (location-related services)

Increasingly, mobile devices are being used as a navigation device as well as to find "information" based on location. Many applications increasingly rely on location data provided through GPS capabilities in mobile devices. From Facebook to yelp, the user's location is being used to facilitate user experience. This has a downside, aside from privacy implications. Location information can be used to launch targeted attacks or associate users' activities based on their location data.

Disabling the GPS is one way to mitigate the risk. However, this is not possible for BYOD (Miller) devices. Another possibility is to educate users on the implications of using location data. Policies preventing some applications (e.g., social media applications) to use location information can also be implemented through policy enforcement.

Lack of control of patching applications and OS

This is an especially acute problem in BYOD environments. Users can bring their own devices and may not patch or update their OS/applications for security fixes that become available, thus exposing enterprise resources to security risks. Think of all the different Android versions (from 2.2.21 to 4.x) in your environment today and the potential security risks for each of them. Users probably have not upgraded or kept up-to-date with security fixes for Android itself. In addition, many users don't install application updates.

Monitoring the devices and trying to ascertain information about the respective versions of their OS/applications can provide information that can be used to flag out insecure OS/applications. Users can then be forced to either upgrade or risk losing access to enterprise resources.

End-user awareness

Any strategy for securing mobile devices or enterprise resources being accessed through mobile devices must include end-user training. Users should be made aware of the risks (listed above) and understand why security controls are necessary. Adhering to these controls should be part of acceptable-use policy, and users should be required to review this at least annually. In addition, annual security-awareness training and a follow-up quiz might imbibe some of these best practices in their minds. Secure awareness should be complemented by warning users when they are about to perform an unwarranted action (e.g., access unwanted site, download malicious code, etc.).

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 (Six; Burns; Chang et. all). Security practices for mobile devices can be divided into four main categories:

1. **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.

2. **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.
3. **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).
4. **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 chapter, 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 chapter.

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.

Other settings

In addition to the above settings, it is strongly recommended to disable the "Make passwords visible" option. This will prevent shoulder surfing attacks, as characters won't be repeated back on screen if the user is typing a password or PIN.

It is also recommended to disable "Allow installation of apps from unknown sources." It was mentioned before, there are secondary application stores apart from Google Play, and it is prudent to not trust

applications from these sources before ascertaining their authenticity. Disabling this option will prevent applications from being installed from other sources.

As a rule of thumb, it is recommended to turn off services that are not being used. A user should turn off "Bluetooth," "NFC," and "Location features" unless using them actively, as well as the "Network notification" feature from the Wi-Fi settings screen. This will make the user choose a connection rather than connecting to any available network. Discourage backing up of data to "Gmail or Google" accounts or Dropbox. Create a whitelist of applications and educate users on the list so they do not install applications outside of the approved list.

A new feature of Android 4.2 enhances protection against malicious applications. Android 4.2 has a feature that, if enabled, verifies an application being installed with Google. Depending on the risk of the application, Android warns users that it is potentially harmful to proceed with the installation. Note that some data is sent to Google to enable this process to take place (log, URL, device ID, OS, etc.).

Another useful feature might be to enable "Always on VPN." This prevents applications from connecting to the network unless VPN is on. Another recommendation is to turn off the USB debugging feature from phones. USB debugging allows a user to connect the phone to an adb shell. This can lead to the enumeration of information on the device. Browser is one of the most commonly used applications on Android devices. Browser security and privacy settings should be fine-tuned (e.g., disable location access).

Android 3.0 and later have the capability to perform full-disk encryption (this does not include the SD card). Turning this feature on encrypts all data on the phone. In case the phone is lost or stolen, data cannot be recovered because it is encrypted. The caveat here is that the screen lock password has to be the same as encryption password. Once the phone is encrypted, during boot time you will be required to enter this password to decrypt the phone.

CONCLUSION

In this chapter 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 (Android and iOS) 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. 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.

The number of daily activated mobile devices rapidly grows and lot of people using these devices every day. These users mainly go to the internet and the use these devices more than classical computers or notebook. From this point of view this is interesting segment for attackers. They know that users have the personal and private data on these devices. The bank segment is trying to move the payment from

computers into the mobile devices with using the NFC technology. This will be discussed topic about the security.

REFERENCES

Backes, M., Gerling, S., & von Styp-Rekowsky, P. (2011). A Local Cross-Site Scripting Attack against Android Phones. *2011. https://www.infsec.cs.uni-saarland.de/projects/android-vuln/android_xss.pdf*

Ballagas, R., Rohs, M., Sheridan, J. G., & Borchers, J. (2004, September). Byod: Bring your own device. In *Proceedings of the Workshop on Ubiquitous Display Environments, Ubicomp* (Vol. 2004).

Bhavani, A. B. (2013). Cross-site Scripting Attacks on Android WebView. *arXiv preprint arXiv:1304.7451*.

Burnette, E. (2009). *Hello, Android: introducing Google's mobile development platform*. Pragmatic Bookshelf.

Burns, J. (2008). Developing secure mobile applications for android.

Chang, G., Tan, C., Li, G., & Zhu, C. (2010). Developing mobile applications on the Android platform. In *Mobile multimedia processing* (pp. 264-286). Springer Berlin Heidelberg.

Clarke, J. (Ed.). (2012). *SQL injection attacks and defense*. Elsevier.

Enck, W., Ongtang, M., & McDaniel, P. D. (2009). Understanding Android Security. *IEEE security & privacy*, *7*(1), 50-57.

Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. (2011, August). A Study of Android Application Security. In *USENIX security symposium* (Vol. 2, p. 2).

Franke, D., Elsemann, C., Kowalewski, S., & Weise, C. (2011, October). Reverse engineering of mobile application lifecycles. In *Reverse Engineering (WCRE), 2011 18th Working Conference on* (pp. 283-292). IEEE.

Halbronn, C., & Sigwald, J. (2010). iPhone security model & vulnerabilities. In*Proceedings of Hack in the box sec-conference. Kuala Lumpur, Malaysia*.

Hoog, A. (2011). *Android forensics: investigation, analysis and mobile security for Google Android*. Elsevier.

Hoog, A., & Strzempka, K. (2011). *iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices*. Elsevier

Johari, R., & Sharma, P. (2012, May). A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on* (pp. 453-458). IEEE.

Kodeswaran, P., Nandakumar, V., Kapoor, S., Kamaraju, P., Joshi, A., & Mukherjea, S. (2012, July). Securing enterprise data on smartphones using run time information flow control. In *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on* (pp. 300-305). IEEE.

Li, J., Gu, D., & Luo, Y. (2012, June). Android malware forensics: reconstruction of malicious events. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on* (pp. 552-558). IEEE.

Li-ping, D. I. N. G. (2012). Analysis the Security of Android. *Netinfo Security*, *3*, 011.

Luo, T., Hao, H., Du, W., Wang, Y., & Yin, H. (2011, December). Attacks on WebView in the Android system. In *Proceedings of the 27th Annual Computer Security Applications Conference* (pp. 343-352). ACM.

Miller, K. W., Voas, J., & Hurlburt, G. F. (2012). BYOD: security and privacy considerations. *It Professional*, *14*(5), 0053-55.

Misra, A., & Dubey, A. (2013). *Android security: attacks and defenses*. CRC Press.

Oh, H. S., Kim, B. J., Choi, H. K., & Moon, S. M. (2012, October). Evaluation of Android Dalvik virtual machine. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems* (pp. 115-124). ACM.

OWASP, T. (2010). 10 2010. *The Ten Most Critical Web Application Security Risks*.

Six, J. (2011). *Application Security for the Android Platform: Processes, Permissions, and Other Safeguards*. " O'Reilly Media, Inc.".

Tesfay, W. B., Booth, T., & Andersson, K. (2012, June). Reputation Based Security Model for Android Applications. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on* (pp. 896-901). IEEE.

Tull, C. (2002). *WAP 2.0 Development*. Que Publishing.

Wei, X., Gomez, L., Neamtiu, I., & Faloutsos, M. (2012, April). Malicious android applications in the enterprise: What do they do and how do we fix it?. In*Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on* (pp. 251-254). IEEE.