CHAPTER 1

Top Mobile Issues and Development Strategies

A discussion on mobile application security must address the current issues facing mobile devices and the best way to mitigate them. This chapter aims to directly provide content on the following subjects:

- Top issues facing mobile devices
- Tips for secure mobile application development

The issues covered in this chapter are not exhaustive and appear in no particular order; however, they can be used to begin the conversation on mobile application security in your organization.

Top Issues Facing Mobile Devices

For any computing device that contains sensitive information and accesses the Internet, security is a major issue. In the 1980s, security issues were hardly noticed; however, security is a major issue for users today (especially for the enterprise user), which includes mobile devices. Let's discuss the top issues facing mobile devices in terms of security. As mentioned, this list of issues is neither exhaustive nor in order of priority; it's just a list of some of the most current issues facing mobile devices and their applications.

Physical Security

Mobile phones will get lost or stolen, period. Whether it is a personal handset or one issued by an employer, the fact that a mobile phone will eventually land in someone else's hands is a security issue. The best-case scenario for a lost/stolen mobile device is a $200 or $300 loss of hardware, but the worst-case scenario is a goldmine of information sitting on the phone (usually in an e-mail inbox) falling into the wrong hands. Furthermore, the fact a mobile device will be lost or stolen is simply one use case—what about the other use case where a user allows another person to borrow their mobile device for a quick phone call? This quick handover equates to an anonymous third party being granted temporary access to a device that holds very sensitive data about the owner (or their employer). How long would it take to download malware on the phone? Probably less time than to make a fake phone call.

In the desktop and laptop world, physical security has always meant no security. The statement is even true to this day, where the latest versions of Microsoft's desktop operating system still seem vulnerable to the very old NT boot disk password-change attack, which requires physical access to change the admin password. Unix environments have had this issue as well, where a user can simply boot into single-user mode and change the root password (which also requires physical access to the machine). The fact that physical access to a device no longer means breaking into data centers or bypassing building security barriers is tough for the IT world because historically it's just a matter of time before someone is able to break through any physical security measures to access data on a disk (it should be noted that on some mobile devices it is the expandable memory slot, such as the MicroSD, that holds all the sensitive data). The best solution for this problem is to design systems assuming physical access will be granted to untrusted parties, and not to assume that any physical security layer will stand the test of time. Unlike many other computing environments—from dedicated servers to cloud computing—physical security will be the number-one issue facing mobile devices.

Secure Data Storage (on Disk)

Securing data on disk relates closely to the previous issue, which is the physical loss of a mobile device. As with laptops, the loss of a mobile device will be a non-issue if the data stored on that device is inaccessible to unauthorized parties. In addition to sensitive documents, information on many mobile applications is stored locally, including password files and authentication tokens, which all need to be protected as well. The ability to store sensitive information locally in a secure manner, and also to keep it accessible to the applications that need it to function properly, is an important requirement for secure mobile computing.

Strong Authentication with Poor Keyboards

Strong authentication—which is defined as a password or passphrase that uses a combination of letters (one of which, at a minimum, should be uppercase), numbers, special characters, and a space—is now the industry standard; however, trying to use that same standard on a mobile keyboard is difficult, if not impossible. The need to uphold strong authentication requirements is imperative, especially if the access is to sensitive data (such as one's bank account); however, enforcing those standards on a mobile keyboard makes even the most paranoid security professional rethink their password strategy.

Multiple-User Support with Security

Traditional client operating systems support multiple users; however, their architectures grant each user a different operating environment. For example, a desktop operating system will require a separate username/password for each user logging into the machine, thus ensuring the data from one account is not readily available to the other. On a mobile device, the world is different. There is no such thing as logging into a mobile device as a separate user (not yet anyway). After entering a four-digit PIN, the user is logged into the system. In this situation, if one application is used purely for business purposes, and the others are personal applications for the family to use, there is no distinction from one application to the next. Each application might need a different security model so the data from one does get exposed to the other; however, because there is one user profile, the device may or may not to be able to support the distinction.

Safe Browsing Environment

One of the biggest exposures to a mobile device is the user's browsing behavior. Many technical issues could be addressed here, but one of the basic issues is the lack of display space on the mobile device. The lack of real estate on a mobile device simply makes a phisher's life easier. For example, the inability to view an entire URL on a mobile browser, or in some cases the inability to view the URL at all, makes all those phishing links significantly more effective. Furthermore, the fact that links are followed a lot more on mobile devices (such as a link from an e-mail or text message) also makes a scammer's life a lot easier. For example, the use of social networking sites on mobile devices combined with the heavy reliance on URL links make it next to impossible for the average user to determine which links are safe and which are not. The mobile browser security model for each device will have to pay special attention to such common but burdensome issues.

Secure Operating Systems

Securing an operating system is no easy task, but it is a task that every mobile software vendor needs to undertake. The task is difficult due to all the constraints mentioned in this discussion, but how well the mobile device vendors address security issues will directly translate to a strong user experience. For example, security often correlates to data loss, but it can also correlate to system downtime. If the lack of strong security prevents a user from even making a simple phone call on their mobile device, the user experience will be tremendously weakened.

Application Isolation

Using a mobile device to make telephone calls is probably the primary reason for the handset; however, the applications installed on the device are a very close second. In most cases, the make-up of the installed applications differs drastically—from corporate applications that support the workplace, gaming applications for entertainment, children's applications to occupy the most demanding members of the family, to social applications to connect to friends and family. These applications, and often the people who use them, require access to different types of data. The ability to isolate these applications and the data they require is an important step in ensuring a simple gaming application does not have access to spreadsheets for a corporate application on the system. A good example of this use case is the old file-sharing programs popular in the early to mid 2000s. Programs such as eDonkey and Limewire gained a lot of popularity because of their rich video and audio access. Because of their peer-to-peer architectures, users were encouraged to share videos and audios file by accessing each other's hard drive, not using a centralized server. The problem with this model is that many eDonkey and Limewire users shared out their entire hard drive, so any corporate information residing on the machine was being shared with all eDonkey and Limewire users as well, thus creating a less than ideal security situation. Refer to the section "Leverage the Permissions Model Used by the OS," later in this chapter, concerning what the different mobile operating systems offer in terms of protection and isolation.

Information Disclosure

Information disclosure has come up many times in this discussion, but it deserves its own dedicated category. The fact that the data stored on the device is worth more than the device itself is nothing new to computing devices—the same idea is true for laptops, desktops, and servers as well. The game changes by the fact that a mobile device has a high likelihood to be lost, stolen, or simply to be used by someone other than the primary owner, which is newer territory for many IT organizations. Furthermore, the loss of data residing on the device is one area of concern, but access from the device to other networks is another. For example, many mobile devices may grant virtual private network (VPN) or extranet access to corporate networks. If this access does not offer a strong form of authentication, then one tiny device falling into the wrong hands could expose the organization's internal network, or at least certain parts of it. This is a significant issue to address and mitigate.

Virus, Worms, Trojans, Spyware, and Malware

As with any device that accesses the Internet, the threat of mobile viruses, worms, Trojans, spyware, and malware needs to be addressed. The good news is that mobile device developers have years of knowledge to leverage from the desktop world; however, with any new computing environment, new attack classes will emerge. The ability to learn from past mistakes is quite important; however, the ability to adjust to current and new threats from viruses, worms, and Trojans will likely be the bigger accomplishment. For example, previous worms that spread through SMS messages and Bluetooth connections are definitely a new attack class, even if they used traditional concepts.

Difficult Patching/Update Process

Patching and updating a mobile device is not a challenge technically; however, other considerations make this process a bit of a problem. The first big hurdle is the mobile carriers. Carriers have big problems with immediate system updates and patching because they have little response time for testing. If a mobile operating system patch breaks four applications running on top of it (which would be nothing new to the patching world), the carriers will be held responsible by their users. For example, if an LG phone is running a custom Android OS and a patch needs to be released, LG will be asked to coordinate with the carriers for a proper release cycle. This may be T-Mobile, Sprint, or AT&T. If any of the carriers see that their user base is being affected negatively, they will probably want to prevent the patch from being deployed quickly, even if it poses a significant security risk. This issue, although not a technical challenge, presents a process challenge that regular desktop operating systems never have to deal with.

Strict Use and Enforcement of SSL

Secure Sockets Layer (SSL) is imperative for a safe and secure operating environment. One does not need to read this book to understand that. This fact is quite obvious by now…or is it? Mobile devices throw us another curveball when it comes to SSL. The first issue is the device itself. Many older versions of mobile devices did not have the computing horsepower to enforce SSL without affecting the highly desired quick/friendly user experience. Although the horsepower of mobile devices has come a long way, older versions of these devices have created the age-old problem of backward compatibility. This means that many mobile applications punted on the strict use of SSL because many devices could not support it well, so they chose not to enforce it at all. Because those versions are still out there, they are still being used. If they are still being used, product managers will hesitate to redirect users to SSL versions of their site because this might reduce the total user base. In addition, other communication from the mobile device to the endpoint system does not always use SSL either. For example, many organizations are defaulting to clear-text protocols for everything, assuming the increased complexity of sniffing on a 3G network is a high enough barrier to punt on SSL; however, as we all know, this is an unsafe assumption that will be proven wrong. Another reason is the abundance of transitive networks between the mobile device and the end system. For example, although the WAP gap days are over (see Chapter 8 for more details), many carriers use proxy services for their mobile clients, storing all the logs on servers they control, not the destination system.

Phishing

Phishing (http://en.wikipedia.org/wiki/Phishing) is a problem on mobile devices. There are many reasons for this, but the main reason is that users need/want to click on items on their phones without thinking about it. Furthermore, many mobile HTML browsers do not even show the full URL of the source web page, thus making a phisher's life easy. There are many articles and whitepapers about this topic, so we'll skip the in-depth conversation about it here. Just note it as a large concern.

Cross-Site Request Forgery (CSRF)

CSRF is an attack class that normally affects web applications (see www.isecpartners.com/documents/XSRF_Paper.pdf for more information). It basically allows an attacker to update a victim's information, such as address, e-mail, or password, on a vulnerable application. For the attack to work, the victim usually has to click on a link that eventually sends them to the destination of their choice, such as a news story about some hot topic, but it also sends many hidden web requests to another application the user happens to be logged into as well, such as a financial application (without the user's knowledge or approval). The attack becomes a big problem for mobile HTML sites that are vulnerable, because mobile users have almost no choice but to click on links from web pages or e-mails to use their phones effectively. The luxury of dissecting a link and its roots is not so easy when one is trying to receive e-mails, send text messages, or browse the Web while operating a vehicle (in some cases). This attack class is healthy in the web application world, but is definitely not dominant or widespread; however, in the mobile HTML world, the attack class will surely be more widespread because its success ratio should be a lot higher. (See Chapter 8 for CSRF testing details on mobile HTML browsers.)

Location Privacy/Security

Privacy is one of those things that is hard to pinpoint with users. All mobile users want privacy; however, a great deal of them will give it away by using products such as Google Latitude. Although Latitude allows a user to control who they share their location with, the idea that someone could know that the user is three blocks away from the nearest Starbucks, has purchased eight Chai teas within the last four days, and will probably buy another cup if they were to get a well-timed coupon in their e-mail might not be too far away. The loss of location privacy is almost a moot point because most mobile phone users have assumed their location privacy was lost as soon as they started carrying a mobile device. Although this may or may not be true, the use of a GPS, location software, or simply one's Facebook page to alert friends about one's whereabouts introduces a new level OS security issues that has never really been a concern for desktop and most laptop operating systems.

Insecure Device Drivers

Although the application layer is generally where users install items, most of those applications should not have *system* access to the device, if the framework architecture was designed correctly. On the other hand, device drivers for mobile devices, such as Bluetooth and video drivers, will need full access to the system in order to perform their functions properly. Although users will not be downloading device drivers on a weekly basics, any device driver that has not been secured properly could be an attack vector, and Achilles heel, for the underlying OS. For example, many mobile operating systems have built in a variety of strong security protection schemes again system-level access to the OS; however, if third-party drivers provide a method to get around these protection schemes via their potentially insecure code, the device will be exposed to attackers. Furthermore, although it may be a poorly written Bluetooth driver that allows an attacker to get system access on a phone, the manufacturer of that phone (Motorola, Apple, HTC, LG) or the operating system vendor (the iPhone, Google, Microsoft) will be assumed guilty by the user, not the vendor of the device driver.

Multifactor Authentication

Multifactor authentication (MFA) on mobile devices is strongly needed. Unlike the PC, a mobile device can fall into the hands of any person, either on purpose (loaned out to make a phone call or look up something on the Internet) or accidentally (a lost or stolen phone). In order to address this issue, many mobile web applications have built in soft multiple factor authentication that is invisible to the user. Most of the soft forms for MFA can be spoofed by attackers, such as authenticating users via the use of a similar browser, the same source IP range, and HTTP headers. With all these solutions, if a phisher is able to collect a user's credentials, the attacker will be able to replay the user's browser information to the mobile web application and pose as the user despite the MFA.

Without thick client mobile applications, it is tough to truly multifactor authenticate on mobile web applications in order to uniquely identify a mobile device. To mitigate this issue, many mobile web applications attempt MFA by creating a device signature associated with the user's mobile phone. The device signature is a combination of HTTP headers and properties of the device's connection. Each time a user attempts to log in, the device signature will be recomputed and compared to the value stored within the mobile web application's database. If the signature does not match, the user must complete a challenge sequence involving out-of-band confirmation through e-mail, SMS, or a phone call. In order to carry this out, mobile web applications are forced to calculate the device ID from information guaranteed present in every request and cannot rely on any stored information from the phone. At a minimum, a device ID will include the first octet of the user's originating IP address and the User-Agent and Accepts HTTP headers. More header information, such as Screen Size, will be included if present; however, not all devices send these headers. HTTP headers are common across all instances of a browser, and header information is easily spoofed. To determine which header values to send, attackers can easily purchase databases of browser header information from Internet sellers or they can harvest the values as part of a phishing site collecting user credentials.

With no device storage and little information available, mobile web applications are incapable of providing robust MFA. A simple penetration test will reveal how easy it is to modify the HTTP headers of an unauthorized browser and masquerade as an authorized browser to access a user's account. Furthermore, by proxying through a server hosted within the source IP range as the target, the attacker would be able to successfully log into the target user's account on a different mobile device (assuming credentials have been compromised as well).

Tips for Secure Mobile Application Development

So how does one write a mobile application in a secure fashion? The answer depends on the platform (Android, iPhone, BlackBerry, Symbian, JME, WinMobile). However, certain basic and generic guidelines apply to all of these. This section provides a short presentation of the best practices for mobile application development. The in-depth details and specific recommendations for each area are discussed in the respective chapters of this book.

Leverage TLS/SSL

The simplest and most basic solution is often the best. Turning on Transport Layer Security (TLS) or Secure Sockets Layer (SSL) by default and requiring its use throughout an application will often protect the mobile device and its users in the long run. Furthermore, both confidentiality and integrity protections should be enabled. Many environments often enforce confidentiality, but do not correctly enforce integrity protection. Both are required to get the full benefits of TLS/SSL.

Follow Secure Programming Practices

To date, most mobile applications are written in C, C++, C#, or Java. If those languages are being used by a mobile development organization, the developers should leverage years of research and use secure programming practices to write secure code. As with any new technology, there is a big rush (and a small budget) to get a product out the door, forcing developers to write code quickly and not make the necessary security checks and balances. Although this scenario is understandable, an abundance of security frameworks and coding guidelines is available. Leveraging these frameworks and guidelines will prevent the security team from slowing down the development cycle and still make the code as safe as possible, preventing the same development mistakes that were made in 1995 from occurring again.

Validate Input

Similar to the preceding topic, validating input is a standard recommendation from most security professionals. Whether it's a full/installed application for a mobile platform or a web application written specifically for a mobile browser, validating input is always imperative. The importance of validating input from full/installed applications on mobile devices cannot be understated. The PC world has lots of host-based firewalls, intrusion detection systems, and antivirus products, but most mobile devices do not have any of these. The situation is similar to plugging a Windows 98 machine into a DSL/cable modem back in the late 1990s. A Windows 98 operating system, and any of the applications running on it, were literally sitting out there on the network for any attacker to target. Although gaining access to the network interface on a mobile device is much more difficult than in the world of DSL/cable modems and Windows 98, the basic sanitization of input is required to ensure any listening services or remote procedure call (RPC) interfaces are not going to crash—or even worse, allow remote control if malformed data is sent to them.

Leverage the Permissions Model Used by the OS

The permission model used by most mobile operating systems is fairly strong on the base device. Although the permissions on the external SD card are usually supported using only the FAT permission model, which is not secure, the base device is well supported by most mobile operating system vendors. For example, the new permission models used by Android and iPhone, where the applications are fairly isolated from each other, should be leveraged as much as possible. There's always the lazy desire to grant a given application access to everything on the mobile device, which is the old Windows 98 way of thinking, but as we saw with Windows XP, that model does not work very well. Although it is easier to create an application that is granted access to the entire OS (rather than taking the time to figure out which services, binaries, files, and processes it actually does need to function), the security architecture of mobile devices will not let the application have such access so easily (see operating system chapters, such as 1–5, for more details). Leveraging the permission model by the mobile operating system will ensure the application plays by the rules.

Use the Least Privilege Model for System Access

Similar to permission models, the least privilege model should be used when developing an application on a mobile device. The least privilege model involves only asking for what is needed by the application. This means that one should enumerate the least amount of services, permissions, files, and processes the application will need and limit the application to only those items. For example, if an application does not need access to the camera on the phone, it should not grant itself access to the process that controls the camera. The least privilege model ensures the application does not affect others and is run in the safest way possible.

Store Sensitive Information Properly

Do not store sensitive information—such as usernames, passwords, or anything else considered sensitive—in clear text on the device. There are many native encryption resources on the major mobile devices, including the iPhone and Android, that should be leveraged. (See Chapter 2 (Android) and Chapter 3 (iPhone) for more details.) Both platforms provide the ability to store sensitive information in a non-clear-text fashion locally on the system. These features enable applications to store information properly on the device without needing third-party software to implement the functionality.

Sign the Application's Code

Although signing the code does not make the code more secure, it allows users to know that an application has followed the practices required by the device's application store. For example, both Apple's iPhone and RIM's BlackBerry have specific processes and procedures that must be completed before an application is published through their stores, which often requires application signing. Furthermore, if an application is going to perform in some sensitive areas, such as needing full access to the device, the appropriate signatures are required to complete these actions. In some cases, if the application is not signed at all, it might have a much reduced number of privileges on the system and will be unable to be widely disturbed through the various application channels of the devices. In some cases, there could be no privileges/distribution at all. Basically, depending on whether or not the application is signed, or what type of certificate is used, the application will be given different privileges on the OS.

Figure Out a Secure and Strong Update Process

This recommendation mirrors the discussion in the "Difficult Patching/Update Process" section, earlier in the chapter. A secure update process needs to be figured out. Much like in the desktop world, an application that is not fully patched is a big problem for the application, the underlying OS, and the user (just ask Microsoft, Adobe, and other large software makers). The idea is to create a process where an application can be updated quickly, easily, and without a lot of bandwidth. Although this may seem like a small issue, it can balloon into a big one if updating software after it lands on a mobile device proves difficult.

Understand the Mobile Browser's Security Strengths and Limitations

If you are writing an application that will leverage the mobile browser, you need to understand its security strengths and limitations. For example, you should understand the limitations of cookies, caching pages locally to the page, the Remember Password check boxes, and cached credentials. Essentially, do not treat the mobile browser as you would treat a regular web browser on a desktop operating system. Understand what security guarantees the mobile browser can provide the web application and then plan appropriately for any gaps. For example, many mobile applications have "read-only" support, meaning the user can access the mobile HTML page of the application, but can only view its contents. The user cannot add, update, or delete any information in this site. Although this is sure to change, it was probably designed with the fact in mind that many mobile browsers and their devices are wildcards right now, so anything more than read-only access would pose too high a security risk. This topic is discussed further in Chapter 8.

Zero Out the Nonthreats

A security book usually is filled with pages and pages of threats, exploits, and vulnerabilities. For each of these issues, it is important to understand the risks to a given technology or topic. In the midst of all those pages, however, it is often overwhelming for many readers who may be lost between "the sky is falling" and "nothing to see here—move along, move along." Although the threats to mobile devices and their applications are very real, it is important to understand which ones matter to a given application. The best way to start this process is to enumerate the threats that are real, design mitigation strategies around them, and note the others as accepted risks. This process is usually called a *threat model* for your mobile application. The threat model should not be too exhaustive or over-engineered. Instead, it should allow application developers to understand all the threats to the system and enable them to take action on those that are too risky to accept. Although this subject is not the most technical one, it will help in the long term when it comes to writing code or performing penetration tests on a mobile application.

Use Secure/Intuitive Mobile URLs

Many organizations are introducing new login pages optimized for mobile web browsing. Adding additional login pages and domains where users are asked to enter credentials gives the users a confusing experience that could actually be detrimental to anti-phishing education. For example, some organizations are using third parties to host their mobile sites. These sites stem from that third party's domain, not the organization's. Furthermore, some organizations are creating and hosting their own sites but are using different extensions, such as .mobi. For years we have been training people about reading URLs and warning them not to go to unfamiliar pages. However, as the following list shows, new mobile URLs can be confusing to the user.

Traditional URL:

- www.isecpartners.com

Common mobile URLs that could be used:

- isecpartners.mobi
- isecpartners.mobilevendor.com
- mobilevendor.com/isecpartners
- Mobile-isecpartners.com

Ideal solution:

- m.isecpartners.com

To mitigate this risk, host the login page under the base ".com" URL and use an "m." prefix.

Conclusion

Mobile application security represents more than the next wave of technology; it will become the default computing method for many point activities in the not-too-distant future, such as e-mail, online shopping, gaming, and even video entertainment. In fact, in countries such as India and China, the mobile device is the primary computing device in the household, not the personal computer. Unlike traditional application waves, such as Web 2.0, the mobile migration involves new hardware, new software, and new applications. This combination brings a whole new world of security challenges to application developers. Fortunately, the industry can leverage 20 years of security research. The picture is not grim for mobile application security, but it is a new green field for attackers and fraudsters. Like in any new platform, security threats will arise that previously did not exist. Although mobile application developers can capitalize on years of research from the desktop application world, the mobile device will bring a new class of issues to the table.

This chapter was presented in two parts—the first being the overall mobile security problem, represented by the list of security issues that face mobile devices and their applications. The second half of this chapter covered the main best practices for mobile application developers to follow. Each of these best practices will be discussed in depth in their respective chapters of the book. For example, although the use of secure storage is imperative, implementing it on a mobile application for the iPhone will be different than on Android. Therefore, you should refer to the respective chapters for specific details and advise on each platform.

Overall, mobile application security should prove to be an interesting industry to follow, and this book is simply a first toward helping developers along the way.