CHAPTER

13

Enterprise Security on the Mobile OS

Enterprise security on mobile operating systems and its installed applications is imperative. Historically, many users have migrated to mobile devices for a variety of reasons, including corporate users, users with limited to no access to a computer, and users wanting to connect to others for social purposes. Although all three classes are equally important, the one major class of users where security is imperative is the enterprise user. For example, whereas Shalin and Sonia, two college students who use their mobile phone to update their Facebook pages every 30 minutes, might not care so much about security features, Jai and Raina, two corporate executives discussing merger and acquisition (M&A) details, will care tremendously (and probably assume it is already built in). As the mobile device migrates from personal use to use in corporations, the data it holds will be considered sensitive, confidential, or "top secret" (especially if you are President Obama; see www.cbc.ca/technology/story/2009/01/23/obama-blackberry.html?ref=rss). As this migration occurs, security options, features, and applications will need to follow along as well. A good example of the mixing of the two worlds is Apple's iPhone. Not only does every corporate executive have one, so does every 22-year coming out of college. The security requirements for the two types of users are quite different—the compromise of a Facebook address book versus the loss of an M&A spreadsheet mean very different things.

This chapter discusses the enterprise security features, support, and applications available on four major mobile platforms—BlackBerry OS, Windows Mobile, iPhone OS, and Google Android. It should be noted that this chapter is simply a quick summary of the in-depth security features discussed in Chapters 2–5. Here are the key categories that will be discussed:

- Device security options
- Secure local storage
- Security policy enforcement
- Encryption
- Application sandboxing, signing, and permissions
- Buffer overflow protection
- Summary of security features

Device Security Options

In terms of mobile device, a few key security features should always be on. Some of these are obvious, but others are less well known.

PIN

Most or all mobile devices have the ability to enable a four-to-eight digit PIN in order to use the phone (outside of 911 services). You should enable the PIN on your phone, period. It's simple and the first step in securing the mobile device. Furthermore, assuming your phone will be lost or stolen at some point in time (even if you just misplace it for a few hours in a coffee shop), an unmotivated attacker will probably not try to break into the OS if they see a PIN has been enabled (but will rather wipe and sell it). The data on the phone, or the data the phone has access to via local or stored credentials, is probably worth more than the device itself.

Although a four-digit PIN only needs 10,000 attempts to brute-force it, many mobile devices have a time delay after ten failed attempts. For example, if someone has stolen a phone for the data and not the device, they will probably attempt to brute-force the PIN. After ten attempts, there is a time delay between attempts, making the 9,990 attempts take much longer. On at least some mobile devices, there is an additional 90-second penalty for every failed attempt above ten, where attempt 11 would require a pause of 90 seconds, attempt 12 would require 180 seconds, attempt 13 would require 270 seconds, and so on. The time delay will not prevent a successful brute-force attack, but will make it considerably harder and longer to perform. The delay should reach a point where the user who has lost the phone is able to notify the appropriate authorities, who can then remotely wipe the phone of its contents (see next section "Remote Wipe"), leaving the attacker with no data after any potential brute-force attack that has actually been successful. Furthermore, some organizations enforce a policy to immediately wipe a mobile phone after ten failed login attempts. Although this may seem aggressive, if an organization is holding sensitive or regulated data, the policy is probably warranted. Furthermore, many corporate phones are fully synced/backed up by enterprise servers, so restoring the data to a new device is trivial (it often takes 45 to 90 minutes).

With some mobile devices, such as the Apple iPhone, the SIM card also has protection, just not the phone. For example, the SIM card in an Apple iPhone will have a PIN as well. If someone steals the SIM card from a device and puts in into another iPhone (in order to steal its data), they will still be required to enter the correct PIN value. To enable a PIN on a SIM or the passlock on an Apple iPhone, complete the following steps:

1. Select Settings | Phone | SIM PIN.
2. Turn on the SIM PIN option.
3. Enter the current PIN (1111 [U.S.], 0000, or 3436).
4. Select Change PIN.

5. Select Settings | General | Passcode Lock.
6. Enter your four-digit code.

To enable a PIN on a Windows Mobile phone, complete the following steps:

1. Select Start | Settings | Security.
2. Select Device Lock.
3. Enter your four-digit code.

Remote Wipe

The ability to remotely wipe data on a mobile device is imperative, especially if it is a smartphone/PDA and is used for corporate purposes. Not only is the remote wipe capability supported on many major platforms using enterprise software, many third-party organizations sell software to remotely wipe your device as well. One way or another, the ability to remotely wipe data off a smartphone/PDA makes the loss of such a device a lot less stressful.

To remotely wipe a smartphone/PDA using a Microsoft Exchange server, complete the following steps:

NOTE

You must be an Exchange admin to perform these functions.

1. Browse to the Mobile Admin site on your Exchange server (https://<Exchange Server Name>/mobileadmin).
2. Select Remote Wipe.
3. Enter the name or e-mail address of the user whose device you wish to wipe (such as **shalindwivedi.com** or simply **Shalin**).
4. Under the Action column, select Wipe to remotely wipe the information from the mobile device. Note that you can select Delete if you simply want to break the connection between the mobile device and the Exchange server, but not necessarily wipe the data.

If direct push is enabled, the device will be wiped immediately. If direct push is not enabled, the device will be wiped the next time the mobile device attempts to sync with the Exchange server.

Secure Local Storage

The ability to store sensitive information locally in a secure fashion is another imperative security feature for mobile operating systems. For example, many applications that are installed on a mobile operating system require some type of authentication to a remote Internet service. Requiring the user to remember and enter authentication credentials each time they want to use the application becomes cumbersome; however, without authentication, the application has no way to identify which user has signed in. For example, many applications installed on the iPhone, Windows Mobile, BlackBerry OS, and the gPhone actually store login information, such as username and password, locally on the device in clear text. Most of the time, the file is easily accessible in backup files with no encryption or obfuscation of this information. This presents a few problems for the user. First, if the device is ever lost or stolen, the owner's username and password for the application are in clear text for all to see. Second, and probably more importantly, other install applications running on the phone could access this same information. For example, any malicious piece of software installed on the phone, such as malware, viruses, or worms, could access the clear-text file with the username and password and then send it to a remote system controlled by an attacker. Furthermore, whereas the storage of username and password information is probably common, some applications may store more sensitive information, such as credit card information (e-commerce applications) and even medical record numbers (medical applications used on a doctor's PDA). The following section covers the iPhone's solution to the local storage issue.

Apple iPhone and Keychain

The iPhone addresses the need to store sensitive credential information on the local device via the use of the Keychain. The Keychain can be used by iPhone applications to store, retrieve, and read sensitive information, such as passwords, certificates, and secrets. Once invoked by an application, the Keychain service ensures an application is verified to access the Keychain by checking its signature (signed by Apple) before granting permissions. The Keychain takes care of all the key management issues, and the application does not have to do much beyond calling to the service.

One key idea to mention is when an application is not using the Keychain and data is being backup to a personal computer. If an iPhone is backed up to a regular computer, all the data on the iPhone will be stored in the clear on the PC, except for data stored

in the Keychain. Hence, if an application truly wants to protect data on the iPhone, it should ensure the Keychain is being used; otherwise, data will be shown in clear text when it is connected to a regular computer.

Security Policy Enforcement

Managing mobile devices is a tough task for IT groups, but a required one. Unlike many other items in the IT world, a mobile device is not only likely to be lost, stolen, or given away at some point in time, but also very likely to have corporate data on it. This combination presents a difficult problem for IT groups, which need to ensure they have some control over mobile devices (and their data) but also know that users may try to bypass security rules if the barriers are too difficult to reach. For example, during the early 2000s, many organizations actually banned the use of 802.11 wireless access points on corporate networks because they were simply "too insecure." Not be denied of the information super highway without wires, many employees simply set up their own rogue access points in cubes and conferences rooms, creating a worse security picture (not knowing one is insecure versus knowing where your weak security points live). Similarly, banning phones and features will likely create a scenario where the IT groups are unaware of the backdoors without having the ability to monitor them on a weekly basis. It is this author's opinion that one cannot prevent users from embracing newer technologies in the name of security—users will do it anyway in a far less secure fashion, so you should embrace it as strongly as you can. Using the music and motion picture industries as an example, stopping a technology wave is impossible, so embracing it is better than trying to fight it.

On the flip side of the IT groups are the mobile device vendors. Some mobile vendors have made the process of securing mobile operating systems easier, while others have not. Whereas IT groups have to own the problem, mobile vendors sometimes make things worse by creating the problem. For example, many mobile devices target the consumer market more rather than the enterprise market. The consumer market cares more about connecting to MySpace, Facebook, and Twitter quickly rather than remotely wiping a device. Therefore, if a mobile device is targeting the consumer market, the enterprise security features offered will usually be less than optimal. Hence, the phones targeted toward the consumer market may have little or no security options available on them, creating a difficult challenge for the enterprise. For example, a user could be buying a mobile phone for personal reasons (sending pictures to family members), but still use the mail, calendar, and document review features on it. It is not possible to separate these two motives for a single type of user, so having enterprise security support across the board makes the process of protecting sensitive data much easier.

Managing mobile operating systems really means the ability to set security policies on the system. For example, similar to the ability of a Windows administrator to require the use of certain types of passwords while avoiding others, this type of policy control is desired on mobile operating systems. Also in the PC world, a local security policy on a Windows/Unix platform can have over 50 different options; this same idea should be true for mobile operating systems. Having a long list of security options an IT organization can enable/disable will go a long way toward data protection, which is the core pain point for these systems in the first place. To date, only a few mobile operating systems have strong support for enterprise security. These vendors not only have the ability to remotely enforce security policies on mobile devices, but have a long list of polices to enforce as well. A good example is the BlackBerry Enterprise Server (BES). BES not only has the ability to manage devices remotely (which includes many of the topics discussed in this chapter), but also has the ability to set fine-grained security polices in the device, such as the minimum encryption key length to be used on the device. A good reference point for each of the major mobile operating systems/devices and the security options they offer can be found on the following links. Also, be sure to reference Chapters 2–5 for the specific implementation details:

BlackBerry Enterprise Server

- http://na.blackberry.com/eng/deliverables/3801/Policy_Reference_Guide.pdf
- www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/7979/1181821/828044/1181292/1272812/1272762/BlackBerry_Enterprise_Solution_Version_4.1.2_Security_Technical_Overview?nodeid=1272692&vernum=0

iPhone

- https://developer.apple.com/iphone/library/navigation/Topics/Security/index.html#//apple_ref/doc/uid/TP40007378

Windows Mobile

- http://msdn.microsoft.com/en-us/library/ms851423.aspx
- https://partner.microsoft.com/40086942

Android

- http://developer.android.com/guide/topics/security/security.html

The key takeaway here is to review the security policies available for the platform you plan to support and enforce the desired security features. Also, users should apply a lot of pressure on vendors who do not have strong security options available but claim to have enterprise support. Supporting the enterprise does not merely mean having an IMAP/POP3 client with SMTP, but rather having the ability to set strong security policies on the device. The latter should ensure the loss of a mobile phone means only a $200 loss to the organization, and not a press release about a data breach on the company's website.

Encryption

Encryption support for mobile operating systems is imperative. The likelihood of losing a mobile phone far exceeds the possibilities of losing a laptop. Although the amount of sensitive data on a laptop far exceeds that on a mobile device, data stored in corporate e-mail and Microsoft Office provides a goldmine for any thief, no manner what form or amount it comes in. This section covers the encryption options in mobile devices, including full disk encryption, e-mail encryption, and file encryption.

Full Disk Encryption

In the Mac and PC worlds, several solutions are offered for full disk encryption, including a few native ones, even on the OS itself (such as Bitlocker on Windows Vista). Unfortunately, the native options are not as widely available on mobile operating systems, which offer little or no solutions for full disk encryption by default. The current security climate will probably change this in the near future, as mobile operating systems will likely embrace the large corporate user base and the data-protection standards it requires, rather than force users to bypass their security teams by using mobile devices in an insecure manner. However, in the short term, users have limited support for full disk encryption, and must rather rely on file or e-mail encryption only, as discussed in the next two sections.

E-mail Encryption

Outside of full disk encryption, e-mail encryption is probably the next best thing. Eighty-five percent of the contents a user would want to encrypt on their mobile operating system is probably e-mail. Of the remainder, ten percent would be e-mail attachments downloaded to the OS in the form of Word, PDF, and Excel documents and five percent would be the storage of authentication credentials.

Although all or most mobile phones support Transport Layer Security (TLS)/ Secure Sockets Layer (SSL) for transmission security, with HTTP, IMAP/POP3, and SMTP, most of them do not support local encryption of stored e-mail. Encryption for locally stored e-mail is important for several reasons. For example, a user may feel secure that their e-mail is passing public communication channels over a TLS tunnel, but if their device were to be stolen, the downloaded e-mail on the device would sit in clear text and in the hands of a malicious person. The need to encrypt locally stored e-mail is obvious—a lost or stolen mobile device could expose plenty of sensitive information sitting in one's Inbox. Furthermore, the few seconds someone "borrows" your phone to make a call could be enough time for a motivated attacker to forward all the e-mail from your phone to a system they control. Unfortunately, none of the most popular mobile operating systems provide native support for local e-mail. BlackBerry devices do offer the best non-native support via the integration of Pretty Good Privacy (PGP). PGP is a popular e-mail encryption tool used on PCs. Using PGP Universal within a BlackBerry enterprise, users can encrypt the contents of an e-mail similar to how it is performed on a PC. Although the use and integration of PGP Universal on BlackBerry Enterprise Servers is not a quick exercise, it does give the corporate enterprise the option to offer the same level of at-rest security protection of e-mail as in the PC world. In addition to PGP, S/MIME is supported on BlackBerry and Windows Mobile as well.

NOTE

More information can be found on integrating PGP or S/MIME to encrypt the actual contents of e-mail (e-mail at rest, not e-mail in transit) on a local BlackBerry device on the BlackBerry website: http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/7979/1181821/828044/1181292/1272812/1272762/BlackBerry_Enterprise_Solution_Version_4.1.2_Security_Technical_Overview?nodeid=1272692&vernum=0.

File Encryption

The last category we discuss under the encryption umbrella is file encryption. A wider amount of support for file encryption, as opposed to e-mail encryption, is provided from the major mobile operating systems. Specifically, BlackBerry, Windows Mobile 6.1, and iPhone (using Keychain) all natively support local file encryption. Both BlackBerry and Windows Mobile 6.1 seem to offer the most seamless encryption options via the use of their policy servers. For example, the BlackBerry Enterprise Server has an option to enable file-level encryption using options on its policy server. Furthermore, Windows Mobile 6.1 users can encrypt e-mail, calendars, My Document files/folders, and tasks by enabling the On-Device Encryption options on the management server.

Application Sandboxing, Signing, and Permissions

Mobile devices have become similar to PCs, where it's almost less about the underlying operating system and more about the applications running on them. For example, the iPhone is a great product, but the applications that run on top of the iPhone OS bring it a significant amount of appeal as well. Similar to the desktop world, if applications are not under tight security controls, they could do more damage than good. Furthermore, as security controls get tighter and tighter on operating systems, attackers are more likely to develop hostile applications that entice users to download/install them (also known as malware) than to try to find a vulnerability in the operating system itself. In order to ensure applications are only allowed access to what they need, in terms of the core OS, and to ensure they are actually vetted before being presented to the mobile user for download, application sandboxing and signing are two important items for mobile operating systems. This section covers some of the security features available on mobile operating systems to protect applications from each other as well as the underlying OS, including application sandboxes, application signing, and application permissions.

Application Sandboxing

Isolating mobile applications into a sandbox provides many benefits, not only for security but also stability. Mobile applications might be written by a large organization with a proper security SDL (software development life cycle) or they might be written by a few people in their spare time. It is impossible to vet each different application before it lands on your mobile phone, so to keep the OS clean and safe, it is better to isolate the applications from each other than to assume they will play nice. In addition to isolation, limiting the application's calls into the core OS is also important. In general, the application should only have access to the core OS in controlled and required areas, not the entire OS by default. For example, in Windows Vista, Internet Explorer (IE) calls to the operating system are very limited, unlike previous versions of IE and Windows XP. In the old world, web applications could break out of IE and access the operating system for whatever purpose, which became a key attack vector for malware. Under Vista and IE7 Protected Mode, access to the core operating system is very limited, with only access to certain directories deemed "untrusted" by the rest of the OS. Overall, the primary goals of application sandboxing are to ensure one application is protected from another (for example, your PayPal application from the malware you just downloaded), to protect the underlying OS from the application (both for security and stability reasons), and to ensure one bad application is isolated from the good ones.

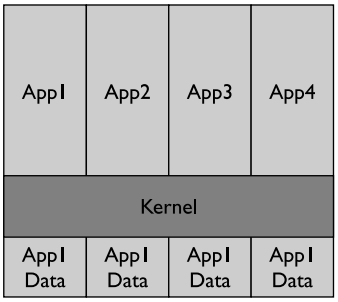

Figure 13-1 *New application isolation model*

All mobile operating systems have implemented some form of application isolation, but in different forms. The newer model of application sandboxing gives each application its own unique identity. Any data, process, or permission associated with the application remains glued to the identity, reducing the amount of sharing across the core OS. For example, the data, files, and folders assigned to a certain application identity would not have access to any data, file, and folders assigned to another application's identity (see Figure 13-1).

The traditional model uses Normal and Privileged assignments, where certain applications have access to everything on the device, and Normal applications have access to the same entities on the device. For example, this model would prevent Normal applications from accessing parts of the file system that are set aside for Privileged applications; however, all Normal applications would have access to the same set of files/folders on the device (see Figure 13-2).

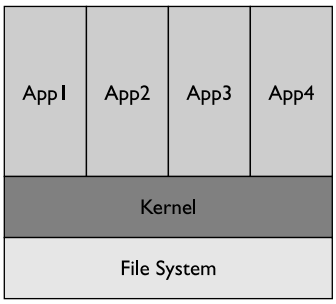

Figure 13-2 *Traditional application isolation model*

The next two subsections provide a short summary of how the different mobile operating systems measure up in terms of application sandboxing (be sure to reference Chapters 2–5 for the specific implementations). Also, much of this information comes from Chris Clark's research on mobile application security, presented at the RSA Conference (https://365.rsaconference.com/blogs/podcast_series_rsa_conference_2009/2009/03/31/christopher-clark-and-301-mobile-application-security—why-is-it-so-hard).

Windows Mobile/BlackBerry OS

BlackBerry devices and Windows Mobile both use the traditional model for application sandboxing. For example, Blackberry uses Normal and Untrusted roles, whereas Windows Mobile uses Normal, Privileged, and Blocked. On Windows Mobile, Privileged applications have full access to the entire device and its data, processes, APIs, and file/folders, as well as write access to the entire registry. Normal applications have access to only parts of the file system, but all the Normal applications have access to the same subset of the operating systems. It should be noted although one Normal application can access the same part of the file system as another Normal application, it cannot directly read or write to the other application's process memory. Blocked applications are basically null, where they are not allowed to run at all.

So how does an application become a Privileged application? Through application signing, which is discussed in the "Application Signing" section. On Windows Mobile, the certificate used to sign the application determines whether the application is running in Normal mode or Privileged mode. If you want your application to run as Privileged instead of Normal, you have to go through a more detailed process from the service provider signing your applications.

iPhone/Android

Both the iPhone and Android use a newer sandboxing model where application roles are attached to file permissions, data, and processes. For example, Android assigns each application a unique ID, which is isolated from other applications by default. The isolation keeps the application's data and processes away from another application's data and processes.

Application Signing

Application signing is simply a vetting process in order to provide users some level of assurance concerning the application. It serves to associate authorship and privileges to an application, but should not be thought of as a measure of the security of the application or its code. For example, for an application to have full access

to a device, it would need the appropriate signature. Also, if an application is not signed, it would have a much reduced amount of privileges and couldn't be widely disturbed through the various application stores of the mobile devices—and in some mobile operating systems, it would have no privileges/distribution at all. Basically, depending on whether or not the application is signed, and what type of certificate is used, different privileges are granted on the OS. It should be noted that receiving a "privileged" certificate versus a "normal" certificate has little to do with technical items, but rather legal items. In terms of getting a signed certificate, you have a few choices, including Mobile2Market, Symbian Signed, VeriSign, Geotrust, and Thawte. The process of getting a certificate from each of the providers is a bit different, but they all following these general guidelines:

1. Purchase a certificate from a Certificate Authorities (CA), and identify your organization to the CA.
2. Sign your application using the certificate purchased in step 1.
3. Send the signed application to the CA, which then verifies the organization signature on your application.
4. The CA then replaces your user-signed certificate in step 1 with its CA-signed certificate.

If you wish your mobile application to run with Privileged access on the Windows Mobile OS, your organization will still have to conform to the technical requirements listed at http://blogs.msdn.com/windowsmobile/articles/248967.aspx, which includes certain do's and don'ts for the registry and APIs. The sticking point is actually agreeing to be legally liable if you break the technical agreements (and being willing to soak up the financial consequences).

The impact to the security world is pretty straightforward, so as to separate the malware applications from legitimate ones. The assumption is that a malware author would not be able to bypass the appropriate levels of controls by a signing authority to get privileged level access or distribution level access to the OS, or even basic level access in some devices. Furthermore, if that were to happen, the application sandbox controls, described previously, would further block the application. A good example of the visual distinction of applications that are signed from applications that are not signed is shown in Figure 13-3, which shows a signed CAB file on Windows Vista (right-click the CAB file and select Properties).

In terms of the major mobile operating systems, most, if not all, require some sort of signing. For example, both BlackBerry and Windows Mobile requiring signing via CAs, although both allow unsigned code to run on the device (but with low privileges).

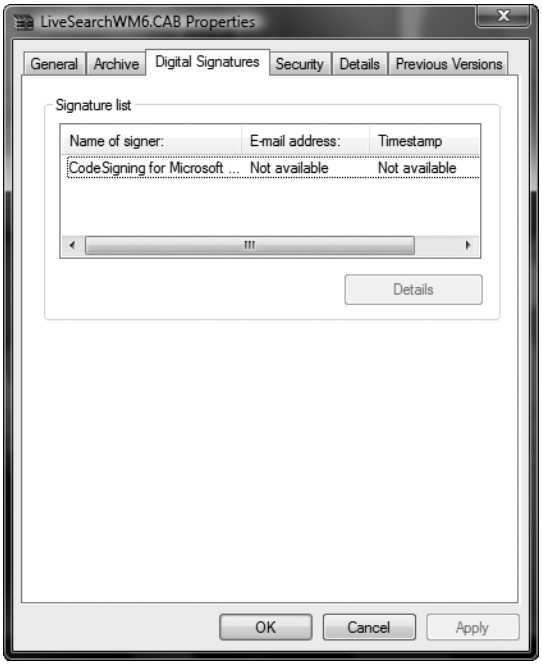

Figure 13-3 *Signed application*

Furthermore, the iPhone and Android require application signing as well, both of which are attached to their respective application stores. Specifically, any application distributed via the application store would have be signed first; however, Android allows self-signed certificates whereas the iPhone does not.

Permissions

File permissions on mobile devices have a different meaning than in regular operating systems, because there's really no idea of multiple roles on a mobile operating system. On a mobile operating system, file permissions are more for applications, ensuring they only have access to their own files/folders and no or limited access to another application's data. On most mobile operating systems, including Windows Mobile and the Apple iPhone, the permission model closely follows the application sandboxing architecture. For example, on the iPhone, each application has access to only its own files and services, thus preventing it from accessing another application's files and services. On the other hand, Windows Mobile uses the Privileged, Normal, and Blocked categories, where Privileged applications can access a file or any part of the file system. Applications in Normal

Data Type	BlackBerry	Windows Mobile 6	Apple iPhone 2.2.1	Google Android
E-mail	Privileged	Normal	None	Permission
SMS	Privileged	Normal	None	Permission
Photos	Privileged	Normal	UIImagePicker Controller	Permission
Location	Privileged	Normal	First Use, Prompts User	Permission
Call history	Privileged	Normal	None	Permission
Secure Digital (SD) cards	Privileged	Normal	N/A	Permission
Access network	Privileged	Normal	Normal	Permission

Table 13-1 *Security Permissions Summary*

mode cannot access restricted parts of the file system, but they all can access the nonrestricted parts collectively. Similar to the iPhone, Android has a very fine-grained permission model. Each application is assigned a UID, similar to the UID in the Unix world, and that UID can only access files and folders that belong to it, nothing else (by default). Applications installed on Android will always run as their given UID on a particular device, and the UID of an application will be used to prevent its data from being shared with other applications.

Table 13-1, created by Alex Stamos and Chris Clark of iSEC Partners, shows the high-level permission model for applications installed on the major platforms for critical parts of the mobile device.

Buffer Overflow Protection

The final category we'll discuss is protection against buffer overflows. Before cross-site scripting dominated the security conversation, buffer overflows were the main attack class every security person worried about. A tremendous amount of good resources for learning about buffer overflows exists. Refer to the following link to get started: http://en.wikipedia.org/wiki/Buffer_overflow.

If an operating system is written in C, Objective-C, or C++, buffer overflows are a major attack class that needs to be addressed. In the case of major mobile operating systems, both Windows Mobile (C, C++, or .NET) and the iPhone (Objective-C) utilize these languages.

The result of a buffer overflow vulnerability is usually remote root access to the system or a process crash, either of which is bad for mobile operating environments. Furthermore, buffer overflows have created serious havoc on commercial-grade operating systems such as Windows 2000/NT/XP; therefore, it is imperative to avoid any similar experiences on newly created mobile operating systems (where most are based on existing operating systems). The main focus of this section is to describe which mobile operating systems have inherited protection from buffer overviews. Because buffer overflows are not a new attack class, but rather a dated one that affects systems written in C or C++, several years have been devoted to creating mitigations to help protect programs and operating systems. The following subsections describe how each major platform mitigates against buffer overflows. Refer to Chapters 2–5 for the specific details.

Windows Mobile

Windows Mobile uses the /GS flag to mitigate buffer overflows. The /GS flag is the buffer overflow check in Visual Studio. It should not be used as a complete foolproof solution to find all buffer overflows in code—nor should anything be used in that fashion. Rather, it's an easy tool for developers to use while they are compiling their code. In fact, code that has buffer overflows in it will not compile when the /GS flag is enabled. The following description of the /GS flag comes from the MSDN site:

> "[It] detects some buffer overruns that overwrite the return address, a common technique for exploiting code that does not enforce buffer size restrictions. This is achieved by injecting security checks into the compiled code."

So what does the /GS flag actually do? It focuses on stack-based buffer overflows (not the heap) using the following guidelines:

- Detect buffer overruns on the return address.
- Protect against known vulnerable C and C++ code used in a function.
- Require the initialization of the security cookie. The security cookie is put on the stack and then compared to the stack upon exit. If any difference between the security cookie and what is on stack is detected, the program is terminated immediately.

iPhone

The iPhone OS mitigates buffer overflows by making the stack and heap on the OS nonexecutable. This means that any attempt to execute code on the stack or heap will not be successful, but rather cause an exception in the program itself. Because most malicious attacks rely on executing code in memory, traditional attacks using buffer overflows usually fail.

The implementation of stack-based protection on the iPhone OS is performed using the NX Bit (also known as the No eXecute bit). The NX bit simply marks certain areas of memory as nonexecutable, preventing the process from executing any code in those marked areas. Similar to the /GS flag on Windows Mobile, the NX bit should not be seen as a replacement for writing secure code, but rather as a mitigation step to help prevent buffer overflow attacks on the iPhone OS.

Android

Google's Android OS mitigates buffer overflow attacks by leveraging the use of ProPolice, OpenBSD malloc/calloc, and the safe_iop function. ProPolice is a stack smasher protector for C and C++, using gcc. The idea behind ProPolice is to protect applications by preventing the ability to manipulate the stack. Also, because protecting against heap-based buffer overflows is difficult with ProPolice, the use of OpenBSD's malloc and calloc functions provides additional protection. For example, OpenBSD's malloc makes performing heap overflows more difficult.

In addition to ProPolice and the use of OpenBSD's malloc/calloc, Android uses the safe-iop library, written by Will Drewry. More information can be found at http://code.google.com/p/safe-iop/. Basically, safe-iop provides functions to perform safe integer operations on the Android platform.

Overall, Android uses a few items to help protect from buffer overflows. As always, none of the solutions is foolproof or perfect, but each offers some sort of protection from both stack-based and heap-based buffer overflow attacks.

BlackBerry

Buffer overflow protection on the BlackBerry OS isn't relevant because the OS is built heavily on Java (J2ME+), where the buffer overflow attack class does not apply. As noted earlier, buffer overflows are an attack class that targets C, Objective-C, or C++. The BlackBerry OS, however, is mainly written in Java. (It should be noted that parts of the BlackBerry OS are *not* written in Java.) More information about BlackBerry's use of Java can be found at http://developers.sun.com/mobility/midp/articles/blackberrydev/.

Feature	BlackBerry	Windows Mobile 6	Apple iPhone 2.2.1	Google Android
PIN	Yes	Yes	Yes	Yes
Remote wipe	Yes	Yes	Yes	No
Remote policy	Yes (BES)	Yes (Exchange)	Yes (Exchange)	No
"LoJack"	Third party	Third party	Not yet	Not yet
Local mail encryption	Yes	No	No	No
File encryption	Yes	Yes	Keychain	No
Application sandbox	Yes	No	No	Yes
Application signing	Yes	Yes	Yes	Yes
Permission model	Fine grained, JME class based	Two tiers	Sandbox, multiple users	Fine grained, kernel and IPC enforced
OS buffer overflow protections	N/A (Java/JME-based OS)	/GS stack protection	Nonexecutable heap+stack	ProPolice, safe_iop, OpenBSD malloc and calloc

Table 13-2 *Security Feature Summary*

Security Feature Summary

We have reviewed a variety of enterprise security features that should be available on mobile operating systems to ensure the security of sensitive/confidential information. During our discussion, we highlighted a few security features required on mobile operating systems and the specific implementations available. However, it would take a full book to cover all of them in the detail they deserve. Table 13-2, researched and created by Alex Stamos and Chris Clark of iSEC Partners, lists all the features we have discussed in this chapter (and a few more), as well as the support level each major mobile device holds for the feature.

Conclusion

Enterprise security features differ from one mobile operating system to the next. Some mobile operating systems are more ready for the enterprise than others in terms of end-to-end security features, but each has its unique benefits. Organizations should not determine which mobile device has the strongest security features and then settle on that one for the entire organization because that view would be too narrow.

Instead, the organization should have a plan ready for each mobile device expected within the enterprise. For example, an organization may determine that the BlackBerry device has the strongest OS from a security perspective and therefore endorse it for the entire company; however, there may be many users within the organization using other devices, such as company executives using the iPhone. Similarly, an organization may think Windows Mobile is the best platform for its users, but then realize its own mobile application is exclusively available through the Android application store, making that platform a required device to support as well.

Realistically, organizations should be prepared for employees to use any of the four major mobile devices, or even a few more, and have a supported security solution for each of them. Although an organization may suggest a supported handset for the enterprise, users will still want what they feel is right for them, even if it is not the preferred solution (as President Obama did in his presidency). The task of supporting several mobile devices in the enterprise is not an easy one. Most organizations don't support four different operating systems for the users' desktops, so supporting four different mobile devices is, quite frankly, an odd but realistic scenario. The refusal to have a security solution for each device expected in the enterprise may mean that corporate data is walking away in an unsupported and uncontrolled fashion, thus making the choice not to support a device much more risky.

This chapter touched on the major enterprise security features currently available on the major mobile platforms. Support for secure local storage, security policy, encryption, and application security are imperative for the enterprise as the mobile phone continues to replace the laptop in usage. Similar to how the laptop replaced the desktop in the enterprise and brought new security challenges along with it, the same thought process will have to take place as the mobile device replaces the laptop for many corporate functions, including e-mail, calendar, application usage, and document viewing.