White paper

Know the big three

Top three mobile application threats

There is no question that mobile computing is growing at an exponential rate. This rapid transformation has seen security concerns outpaced by the ease of use, flexibility, and productivity of mobile devices. When vulnerabilities are exploited, the security of mission-critical data becomes a serious concern. Here we take a look at three of the top mobile application security threats facing businesses today and recommendations on how to mitigate the risk.

What you will learn

- Mobile applications are just as prone to security vulnerabilities as their web counterparts.
- Insecure use of mobile APIs, data exposure in transit and at rest, and other serious threats make this shift to mobile computing a top concern for businesses today.
- The top three mobile application security threats.
- Recommendations on how to mitigate the risk from mobile computing security vulnerabilities.

What you should know

The reader should have a basic understanding of mobile development processes, how mobile applications work, and application vulnerabilities.

About the author

Mark Painter has been in the security industry since 2002 when he first joined SPI Dynamics. During his tenure, he has been involved with security blogging and vulnerability research. He currently serves as the product marketing manager for the HP Fortify WebInspect product suite and the HP Fortify on-demand professional services organization.

The risks of mobile applications

According to Morgan Stanley Research, the smart phone will become the dominant computing platform by the end of 2012, with more units being sold than desktops and laptops combined.[1] It's been a remarkable and rapid transformation that much like the advent of the web has left security concerns outpaced by the ease of use and flexibility of a new tool.

The HP Fortify on-demand manual testing team conducted a recent study to analyze security threats associated with a number of mobile applications and to identify what vulnerabilities are occurring in the wild. Unfortunately, we've found that applications on mobile devices are just as prone to security vulnerabilities as their web counterparts. We've seen numerous instances of insecure use of mobile APIs, data exposure in transit and at rest, and other serious threats. Both iOS applications and Android applications were represented. The majority of the applications were developed by third-party development groups. In terms of numbers and the danger that could arise when the vulnerabilities are exploited, here are the top three security concerns that we discovered in our sample set, along with recommendations as to how to mitigate the associated risk.

1 Source http://voices.washingtonpost.com/posttech/2010/11/smartphone_sales_to_pass_compu.html

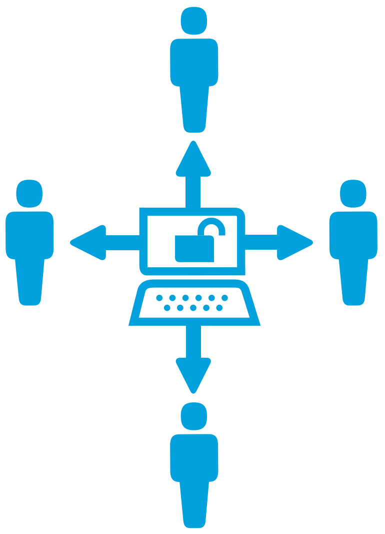

Figure 1
Sensitive data leakage over insecure channels

Sensitive data leakage over insecure channels

Data leakage has been a long-standing issue among web applications. While it can seem low-key, it is often a seemingly innocuous piece of information that lets an attacker escalate his attack methodology to conduct a more dangerous attack. The same is true in mobile applications. Over half the applications (51%) in our survey were susceptible to information leakage vulnerabilities. We discovered that a user's personal data was often sent over unencrypted network protocols such as HTTP. A lot of this information was simple, such as names, addresses, and phone numbers. However, across our sample set this data also included the current location of the user, and the specific device identifier (aka the UDID).

The device identifier is very important in that it can be leveraged for incredibly targeted attacks against specific users. If the geo-location, unique device identifier, and personal details of the device owner could all be intercepted via a vulnerable application, the real-world implications are staggering. It's James Bond stuff, to say the least. An attacker could locate a "target" in the real world and then, well, who's to say what would happen. It's a frightening scenario most people don't imagine. Of course, simple run of the mill application exploitation could also take place. Imagine this scenario. If the application has been sending the UDID, full name, address, etc., to a vulnerable web service, and that web service is susceptible to SQL Injection, then it's easily conceivable every bit of data on that mobile device could be accessed. It's amazing how far information leakage can take an attacker given the right set of circumstances; and none of them out of the realm of the probable, let alone possible.

Data transmitted over insecure channels was not limited to personal data—application data was also not secure. We discovered login information, user credentials, session IDs, tokens, and sensitive company data all being sent over unencrypted network protocols like HTTP. Imagine the consequences for a vulnerable banking application. If credentials, session identifiers, identifiable personal information, or other sensitive data are being transmitted to a backend server, the transmission should be secure. Otherwise, data could be intercepted by an attacker using common network packet capturing tools or apps (for example, DroidSheep).

Applications play loose and fast with data in other ways, too. Seventy-five percent of the applications we tested are capable of sending tracking data to third-party advertising and analytics providers. While not technically a vulnerability, it does offer more attack vectors for a potential attacker if those providers are themselves not secure or sending the data over an insecure connection. Mobile application developers should consider the security of everything their applications can communicate with, not just their own applications, but every third-party service or library they use to build their application.

Although Apple has recently started cracking down on the use of UDIDs by deprecating the API calls for accessing the UDID in recent versions of the iOS SDK, applications built upon older versions of the SDK still make use of UDIDs. As an example, the popular Flurry analytics third-party library sends the UDID over an unencrypted connection to Flurry's web service APIs. Being proactive, Flurry released the following update to app developers that makes use of an alternate identification scheme for a user's device: http://support.flurry.com/index.php?title=Analytics/FAQ/RequiredFlurryiOSUpdate.

Lost/Stolen devices

Devices get lost. Devices are stolen. This is nothing new, but with the proliferation of mobile computing, the lengths that businesses must go to in order to secure the vulnerabilities introduced by lost or stolen devices have become front and center.

Encryption on corporate PCs is now standard protocol for most Fortune 500 companies. Ten years ago the news was rife with stories of data stolen from lost PCs. This has definitely been curtailed in part because of legislative requirements, and in part because corporations have learned their lessons the hard way. However, these same standards are not yet being applied to mobile devices, and in the age of "bring your own device" (BYOD), that's dangerous.

Mobile applications have unique concerns when the device they are running on is lost or stolen—even more so than PCs. Sixty-eight percent of the applications we tested did not secure the data stored on the device. As a result, attackers may obtain elevated privileges on a stolen device to access sensitive application data. It is imperative that any credentials on a mobile device be either encrypted on Android or stored to the keychain on iOS. Application sandboxing (limiting the resources the application can access) and code signing (putting restrictions in place to guarantee the code has not been altered) can help mitigate this in most scenarios. However, these can be bypassed by common device rooting (gaining privileged control) and jail-breaking techniques, which provide the user with total access to the entire file system of the device.

Resources

For service overview, ordering guide and technical overview
hp.com/managedprintservices

Malicious applications

We've spent a lot of time talking about vulnerable applications and how they can be exploited. However, there is a new concern specific to the mobile arena in that applications need to be protected not only from outside agents but also from other applications stored on that device. That's a sea change in terms of what needs to be done to secure applications. Almost a quarter (24%) of the applications we tested logged or stored sensitive data on the device that was readable by other non-privileged applications on the device. Additionally, 10% of the applications we tested allowed attacks via inter-application communication or via weak permissions (Android intents/permissions or iOS custom handlers). Malicious applications can typically only access another application's data if it was stored world-readable (i.e. SD card) or if the application logged any sensitive data (Android log method or iOS NSLog method). If a malicious application is able to load code that can elevate privileges it may be able to totally compromise another application's data. iOS code is signed and under normal circumstances should prevent loading external code. However, code signing has been bypassed in the past. Android currently does not sign code, although it is self-signed, which is not entirely effective.

Inter-application communication can occur in both Android and iOS. Android uses a more granular, complex model than iOS applications to support inter-app communication. While working with developers, we've observed developer confusion around how to properly implement Android permissions as well as inter-app communication due to this more complex model. For Android, developers should use the principle of least privilege and only define necessary permissions in AndroidManifest.xml for the application to function properly. Also for Android, caution should be exercised when sending implicit Intents and exporting components. Explicit Intents should be used when possible. Exporting components should be avoided unless absolutely necessary. For iOS, developers should validate the source bundle identifier to the openURL method when implementing custom protocol handlers. All sensitive logging calls should be disabled for applications in production. And for Android, sensitive data should never be allowed to be written to world-readable/writable files or stored to the SD card.

Recommendations

There are certain actions that organizations can take to mitigate the risk from mobile application security vulnerabilities. First, applications need to be manually audited and assessed before the products are launched to determine if any input injection vulnerabilities or information leakage vulnerabilities are present. The code should be analyzed via static analysis when being developed to find code-based vulnerabilities. As with any application, it's much less expensive to address security vulnerabilities during development than once it has been released.

Secure data transmission standards should be included as part of any application requirements, especially if those applications are being developed by third-party developers. The same goes for secure data storage and application logging. Reasonable inter-application communication exposure and permissions in application requirements should be stringently defined. These concerns should all be addressed in the requirements phase and tested during development.

When performing security testing and analysis on mobile applications, the server-side web services and APIs that the mobile clients talk to should be taken in context and analyzed for vulnerabilities. High-risk vulnerabilities may be missed if the two are tested out of context with each other.

Sign up for updates
hp.com/go/getupdated

Share with colleagues

Rate this document

4AA4-4446ENW, November 2012