Smartphone Security

Lori Flynn and Will Klieber, CERT

Smartphones handle and store sensitive data that should be protected. The vast amount of private information stored on smartphones was even cited by the US Supreme Court, in *Riley v. California* (2014), as a factor in ruling that searches of these devices require a warrant. *Taint-flow analyzers* use static or dynamic analysis techniques to trace the flow of sensitive data to undesired locations.

If a user's location data, such as GPS coordinates or Wi-Fi access point information, is disclosed, it can compromise the user's privacy and, in extreme cases, put the user's physical safety at risk. Medical information is also increasingly an issue, given the increased popularity of wearable computing devices (such as health sensors) that communicate with users' smartphones. In addition, data from the phone's sensors or stored on the device (in emails, texts, or photos) could be used for theft (bank and credit card numbers), blackmail, stalking, unfair competition, public humiliation, and other abuses. Malware could surveil the smartphone user with microphone, video, and other sensors. Furthermore, privacy threats to users can come from many sources, including advertisers, hackers, and governments. Finally, employees often use their smartphones for both personal and business purposes; accordingly, technological measures should ensure that the employee's personal data is not leaked to the employer and that proprietary business data is kept secure.

Here, we discuss in detail various smartphone security issues and present tools and strategies that can help us better protect sensitive data.

Security Issues

Smartphones present a unique environment that comes with its own set of security concerns (see the "Desktops vs. Smartphones Security" textbox for more information).

Desktops vs. Smartphone Security

Under popular desktop operating systems (including Windows, Mac OS X, and Linux), programs usually execute with all permissions of the user. Smartphone apps are more tightly constrained. Apps must request and be granted permission to do things, such as reading from the microphone or accessing the phone's general file system. Apps are sandboxed from each other more tightly than on desktop OSs. On Android, each app has private storage that other apps can't read or write. Unlike desktop programs, which can be run with root privileges via the `su` command or the Windows User Account Control, third-party apps on Android and iOS smartphones can't be run as root unless the user has unlocked the phone's bootloader. App stores perform some checks on apps to try to prevent malicious apps from being released on the app store.

Operating System Vulnerabilities

Each smartphone operating system (OS) has security vulnerabilities particular to its system. For example, Apple and Microsoft have a mechanism to push out security updates to smartphones using their OSs, but Google can only push updates to pure-Android devices, such as Nexus phones.

Google provides fixes to original equipment manufacturers (OEMs) and service providers (SPs) that provide specialized versions of the Android OS, but OEMs and SPs often don't implement and distribute fixes, or take a long time to do so. Recent studies show Android OS updates permeate extremely slowly over Android phones. Only 0.7 percent of Android phones use the latest OS version, while widely fragmented large segments of Android users have old OS versions.[1] Missing security fixes hits lower-cost Android phones the hardest: many receive no updates and others only rarely. This issue recently has been highlighted by the public disclosure of Android Stagefright vulnerabilities, a severe problem that might allow a remote attacker to execute code on Android devices.[2] An estimated 950 million Android phones are still vulnerable,[3] over 3 months after a security researcher disclosed the vulnerability to Google along with code patches, even though Google applied the patches to internal code branches within 48 hours.

Additional OS-specific issues include the following.

iOS security issues. Widespread vulnerabilities have recently been shown in iOS app-to-app and app-to-operating-system communications,[4] involving scheme hijacking and possibly WebSocket abuses. These vulnerabilities are due to a lack of authentication for multiple reasons: iOS doesn't provide some types of authentication APIs, enforce some authentication, or advise developers to check for particular authentications. The Xavus tool found many of these exploitable vulnerabilities in popular iOS apps.[4]

Android security issues. Android has a complex inter-app communication system that can be used in attacks. An *intent* is a message sent to a component of an app. An intent might explicitly designate its recipient by name, or it might rely on the OS to find a suitable recipient by matching properties of the intent to potential recipients' intent filters. The latter type of intent, an *implicit intent*, poses the greatest security concerns.

Intents can be used to make it difficult to statically analyze the flow of sensitive data between apps in a precise manner (that is, with few false negatives and few false positives). *Intent hijacking* occurs when a malicious app receives an intent that was intended for (but not explicitly designated for) another app. If two apps have *activity* components that can handle an implicit intent, then the user is presented with a choice of which app to use. A malicious app can try to trick the user into choosing it by using a confusing name. Also, an inattentive user might not give much thought to the choice. Furthermore, the touchscreen might register a tap for the malicious app that the user did not intend.

Beyond inter-app communication, intents are also used for intra-app communication between different components of a single app. It is easy for a developer to mistakenly make app interfaces public when they should be private, allowing malicious apps to eavesdrop or hijack data. Epicc is a static-analysis tool that analyzes inter-component communication vulnerabilities.[5]

Memory Corruption Attacks

Memory corruption attacks (such as buffer overflows) commonly exploited on desktop systems are also applicable to mobile devices. In Android, many apps are written purely in Java, a memory-safe language, which limits the attack surface to

- apps that employ native code;
- vulnerabilities in the Java virtual machine and the Java runtime environment; and
- vulnerabilities in the underlying OS.

Mitigations include address-space layout randomization (ASLR) and data execution protection (DEP).

DEP allows regions of memory (such as the stack) to be marked with a "non-executable" (NX) bit, which the CPU checks before executing code from the memory region. Partial ASLR support has been present on Android since 4.0 and on iOS since 4.3; however, even small libraries unprotected by ASLR have been shown to offer sufficient gadgets for ROP exploits.[6] Modern Android and iOS versions use DEP on supporting hardware. Return-oriented programming (ROP) is a technique to exploit memory corruption even in the presence of DEP. Rather than writing new executable code onto the stack, the exploit takes advantage of existing *gadgets* (small sequences of machine code that typically end with a `RET` instruction) that can be effectively chained together. A ROP exploit is used by the evasi0n jailbreaking tool for iOS 6.0.

Protective Measures (and Some Failures)

Just as each OS has its own vulnerabilities, each also has security measures specific to its system. Also, some protective security measures need to be applied (and researched), regardless of the OS.

OS-Specific Security

Different smartphone OSs allow varying levels of user control (and protection) over sensitive dataflow. The smartphone OS with the largest worldwide market share, Android, currently offers only limited control by users over their data, requiring all permissions requested to be granted before an app is installed. The public release of the Android M software developer's kit (SDK) is scheduled for Q3 of 2015 (https://developer.android.com/preview/overview.html), and it changes the Android permissions model, so permissions won't need to be requested during installation, can be asked for during use as needed, and can be revoked by users without removing the app.

The M SDK also introduces *App Links*, which enable a website to designate an official app, which, if installed, will automatically be chosen as the default handler for links to that website. This helps mitigate intent hijacking if a malicious third-party app also tries to register itself to handle those links. The M SDK will increase Android security in additional ways, including Wi-Fi, Android application package (APK) validation, camera use, and more.

The second-highest market share smartphone is iOS. In iOS 8, users can install apps and control permissions afterward, although with limited granularity. In contrast to the current Android permissions model, iOS prompts the user to grant permissions only when the app is actually about to use the permission.

The worldwide third-highest-selling smartphone OS consistently (from 2012 through 2015) is the Windows Phone, which in Q1 2015 is estimated at almost three percent of worldwide smartphone sales (see www.idc.com/prodserv/smartphone-os-market-share.jsp). As opposed to iOS and Android, Microsoft provides developers five different application models for building Windows Phone apps. This adds to the complexity of app analysis, as well as to the analysis of dataflow and control (both app-to-app and app-to-system). Microsoft provides a tool for profiling and monitoring some behaviors of apps, and researchers have created some app analysis tools, but the Windows Phone lacks the number and depth of dynamic and static analysis frameworks and tools that exist for Android and iOS apps.

Analysis Tools

Many Android app analysis tools are built on the Soot[7] and T.J. Watson Libraries for Analysis (WALA) static analysis frameworks, and there are many standard dynamic analyzers (such as DroidScope[8]) and fuzzers (such as Droidfuzzer[9]) for Android apps. There are many analysis tools for iOS, including the PiOS[7] and Xavus[11] static analyzers and the PSiOS policy enforcement framework.[12] Static and dynamic (including fuzzing) analysis of potential dataflows and control flows are vital for understanding potential security issues in each smartphone system, including apps.

Moreover, vulnerabilities inherent to programming languages used for the systems should be examined, along with the security of the system in general (including encryption, deletion, password handling, and communications protocols used). Systems with small market shares tend to have fewer analytical tools. Figure 1 shows a high-level view of taint-flow analysis, which can be done with static tools (such as DidFail[13]) and dynamic tools (such TaintDroid[14]).

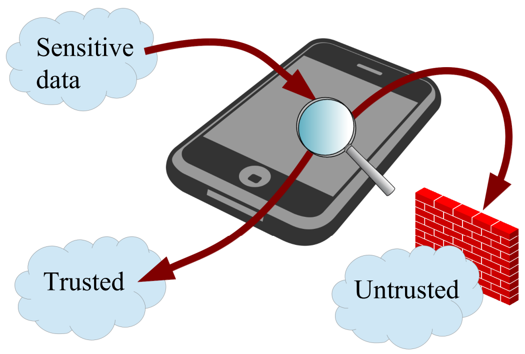


Figure 1: Taint flow analysis can be used in protecting against the flow of sensitive data to undesired locations.

Smaller-Market Phones

Cyanogenmod is an open-source firmware distribution based on Android that lets users install apps without granting all requested permissions. It also lets users substitute fake data instead of real data (for example, in place of real location data). Blackphone has an OS that is based on a fork of Android. It uses peer-to-peer encrypted calling and video, and it can use a privacy-focused enterprise management system. Silent Circle (the maker of Blackphone) has a privacy-focused app store, including Android and iOS apps with full call and text encryption (https://www.eff.org/secure-messaging-scorecard). Additional smartphone OSs with much smaller market shares include Blackberry, Symbian, Ubuntu, and China Operating System (COS).

Vulnerability Coordination

Despite the Blackphone's focus on security, a data-type confusion vulnerability in its code was disclosed and fixed in January 2015. The vulnerability could have allowed remote attackers to execute arbitrary code on Blackphones. This is a good example of how difficult it can be to secure smartphone communications and data, and of the importance of vulnerability report management. Blackphone's website has a secure form for reporting vulnerabilities. OS providers and app creators should have a way for the public to report security vulnerabilities and should work quickly to address them. Bug bounties are incentives to motivate vulnerability disclosures and coordination with developers.

If the reporting method is insecure, a report could be intercepted by a third party, who could use it to exploit the vulnerability.[15] Google Android, Apple iOS, and Microsoft Phone have secure vulnerability reporting, coordination, and rewards programs. App developers might not respond to vulnerability disclosures, so to protect users, reporting should be coordinated by the app stores. CERT also handles vulnerability coordination between reporters and vendors/developers as a free public service.

App Permissions and Languages

Most users do not understand the full implications of allowing app permissions. A study in 2011 by Adrienne Porter Felt and her colleagues found evidence that even many developers don't fully understand permissions.[16] They found that many apps request extraneous permissions that aren't needed by any of the API calls that the app makes. They also found that, in many cases, the Android documentation about permissions was missing or incorrect.

User-experience researchers[17] work to understand effective (and ineffective) methods of conveying information to users who are not technical experts. Similar research projects strive to effectively support secure coding of apps with integrated development environment (IDE) assistance, secure coding standards, and other tools to analyze and improve app security during development. Developer education helps, including secure coding training for particular programming languages and OSs.

Undefined behavior in programming language standards leads to security vulnerabilities. Developers should follow secure coding standards for the programming languages and for the mobile OS, which impose rules and recommendations for coding securely that mitigate problems due to officially undefined behaviors. The smartphone's OS, drivers, application framework, virtual machine environment, and apps can be written in a variety of languages. For example, the Android OS is written mostly in C, runtime libraries are written in C/C++ except the Java Core libraries, and Android apps are written in Java but can incorporate native code (such as C or C++).

Hybrid Apps

Although hybrid Web/mobile application frameworks make development of cross-platform apps possible, recent research has shown serious vulnerabilities that expose sensitive local resources to malicious Web domains,[18] affecting all hybrid frameworks and smartphone platforms that deploy the frameworks.

Cyber-Hygiene

Other factors in smartphone security could be helped by public-education programs similar to public-health education (such as campaigns to promote covering your mouth when sneezing) but for cyber-hygiene. Some users do not have a password login for their phone or a timed lockout, much less security afforded by phone encryption. These basic data protections should be used by everyone, given that devices are often lost or misplaced.

The above basic protections adequately protect data in many cases, but they are not fool-proof. A password-locked phone can be attacked by analyzing the smudges left when entering the password.[19] Sophisticated adversaries might be able to recover encryption keys from a powered-on Android phone's RAM[20] by a method involving physically chilling the phone.

USB power plugs could be abused as a data-channel attack vector against users who think they are simply charging their phone; a mitigation is to use a USB condom when connecting to an untrusted charging outlet. All personal data in the phone should be securely deleted before a user disposes of their phone. Backing up data by syncing it to a local machine or cloud protects the user's access to data even if a device is destroyed or lost, but privacy of the backed-up data depends on the backup system's privacy protections. A cyber-hygiene campaign could make more users aware of these risks and mitigations.

Women's clothing in particular presents a smartphone security issue, because most women's slacks and skirts do not come with front pockets even close to large enough to fit a smartphone. (However, you can have a tailor extend your front pockets to securely carry a smartphone.) Carrying a phone in a purse, backpack, or jacket pocket increases the likelihood of theft or loss, plus the risk of tampering (such as inserting a key logger), compared to carrying it in pants pockets.

Encryption

SSL, if used correctly, promises to provide secure end-to-end communication over an insecure channel. A comprehensive research project, which analyzed Google Play apps that use cryptographic APIs, showed that 88 percent used SSL incorrectly.[21] Tools such as mallodroid and CERT Tapioca find SSL vulnerabilities in apps. Furthermore, a standard Android, iOS, or Windows Phone and browser are vulnerable to a *compelled certificate creation attack,* in which government authorities would compel a certificate authority to issue false SSL certificates for covertly intercepting and hijacking secure Web-based communications.[22]

Cell phones encrypt voice data using keys in SIM cards. However, if an attacker obtains these SIM keys, decryption of phone communications using those SIMs is trivial. Gemalto, which manufactures approximately 2 billion SIM cards annually, was reportedly hacked and its SIM cards' encryption keys were stolen.[23]

Baseband

The *baseband OS* provides another attack surface. Most smartphones include two operating systems on two different processors: the general-purpose applications processor runs the main OS (for example, Android or iOS) and a processor that executes a proprietary real-time OS and manages all radio functions (the baseband OS). Stingray technology uses vulnerabilities in baseband technologies, such as knocking phones off a 3G network and onto an insecure 2G network with a fake base station, to intercept cellphone communications.[24]

Baseband software is currently poorly understood, because it is closed-source. Tools available to the public for analyzing baseband software are limited, and baseband is a promising area for vulnerability research and mitigation. OpenBTS, OsmoBTS, and OpenLTE are open source software that enables software-defined radio communications, making research on mobile baseband security more affordable. Most baseband processors are ARM processors, which the widely used IDA Pro disassembler supports. Google's Bin Diff tool has also been used by baseband researchers to identify and match functions in binaries. Increasingly, research publications detail baseband vulnerabilities and potential attacks that have been researched using OpenBTS with software defined radios, IDA Pro, and Bin Diff.

Summary

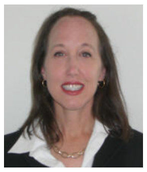
The security landscape of mobile devices is far from ideal, and there are many problem areas ripe for further research. Exciting, high-impact topics for research include better user interfaces, improved encryption, finding and securing baseband OS vulnerabilities, and many more. Non-research work needed includes public cyber-hygiene educational campaigns and improved distribution for security updates.

References

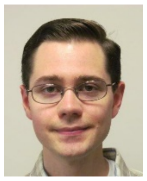
1. L. Armasu, "Google Can't Ignore The Android Update Problem Any Longer," *Tom's Hardware*, 5 May 2015; www.tomshardware.com/news/google-android-update-problem-fix,29042.html.
2. G. Wassermann, *CERT Vulnerability Note VU#924951*, Vulnerability Notes Database, July 2015; www.kb.cert.org/vuls/id/924951#sthash.2Z6iNXBT.dpuf.
3. J. Minor, "There's (Almost) Nothing You Can Do About Stagefright," *PC Magazine*, 30 July 2015; www.pcmag.com/article2/0,2817,2488772,00.asp.
4. L. Xing et al., "Unauthorized Cross-App Resource Access on MAC OS X and iOS," 2015; http://arxiv.org/abs/1505.06836.
5. D. Octeau et al., "Effective Inter-Component Communication Mapping in Android: An Essential Step Towards Holistic Security Analysis," *Proc. 22nd USENIX Conf. Security* (SEC), 2013, pp. 543–558; http://dl.acm.org/citation.cfm?id=2534813.
6. E. Schwartz et al., "Q: Exploit Hardening Made Easy," *Proc. 20th USENIX Conf. Security* (SEC), 2011, p. 25; http://dl.acm.org/citation.cfm?id=2028092.
7. R. Vallée-Rai et al., "Soot—A Java Bytecode Optimization Framework," *Proc. 1999 Conf. Centre for Advanced*

Studies on Collaborative Research (CASCON), 1999, p. 13; http://dl.acm.org/citation.cfm?id=782008.

8. L.K. Yan and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," *Proc. 21st USENIX Conf. Security Symp*, 2012, pp. 569–584.
9. H. Ye et al., "Droidfuzzer: Fuzzing the Android Apps with Intent-Filter Tag," *Proc. Int'l Conf. Advances in Mobile Computing & Multimedia* (MoMM), 2013, p. 68; http://dl.acm.org/citation.cfm?id=2536881.
10. M. Egele et al., "PiOS: Detecting Privacy Leaks in iOS Applications," *Proc. 18th Ann. Network and Distributed System Security Symp*., 2011; https://iseclab.org/papers/egele-ndss11.pdf.
11. L. Xing et al., "Unauthorized Cross-App Resource Access on MAC OS X and iOS," 2015; http://arxiv.org/abs/1505.06836.
12. T. Werthmann et al., "PSiOS: Bring Your Own Privacy & Security to iOS Devices," *Proc. 8th ACM SIGSAC Symp. Information, Computer and Communications Security* (ASIA CCS), 2013, pp. 13–24; http://dl.acm.org/citation.cfm?id=2484316.
13. W. Klieber et al., "Android Taint Flow Analysis for App Sets," *Proc. 3rd ACM SIGPLAN Int'l Workshop on the State of the Art in Java Program Analysis* (SOAP), 2014; http://dl.acm.org/citation.cfm?id=2614633.
14. W. Enck et al., "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *ACM Trans. Computer Systems*, vol. 32, no. 2, 2014, article no. 5; http://dl.acm.org/citation.cfm?id=2619091.
15. A. Fishman and M. Marquis-Boire, "Popular Security Software Came under Relentless NSA and GCHQ Attack," *The Intercept*, 22 June 2015; https://firstlook.org/theintercept/2015/06/22/nsa-gchq-targeted-kaspersky.
16. A.P. Felt et al., "Android Permissions Demystified," *Proc. 18th ACM Conf. Computer and Communications Security* (CCS), 2011, pp. 627–638; http://dl.acm.org/citation.cfm?id=2046779.
17. A.P. Felt et al. "Android Permissions: User Attention, Comprehension, and Behavior," *Proc. Eighth Symp. Usable Privacy and Security* (SOUPS), 2012, article no. 3; http://dl.acm.org/citation.cfm?id=2335360.
18. M. Georgiev et al., "Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks," *Proc. Network and Distributed System Security* (NDSS), 2014; www.cs.utexas.edu/~suman/publications/suman_ndss14.pdf.
19. A. Aviv et al., "Smudge Attacks on Smartphone Touch Screens," *Proc. 4th USENIX Workshop on Offensive Technologies* (WOOT), 2010, pp. 1–7; http://dl.acm.org/citation.cfm?id=1925009.
20. S. Anthony, "How to Bypass an Android Smartphone's Encryption and Security: Put It in the Freezer," *Extreme Tech*, 12 Mar. 2013; www.extremetech.com/computing/150536-how-to-bypass-an-android-smartphones-encryption-and-security-put-it-in-the-freezer.
21. M. Egele et al., "An Empirical Study of Cryptographic Misuse in Android Applications," *Proc. 2013 ACM SIGSAC Conf. Computer & Communications Security* (CCS), 2013, pp. 73–84; http://dl.acm.org/citation.cfm?id=2516693.
22. C. Soghoian and S. Stamm. "Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL," *Financial Cryptography and Data Security*, LNCS, Springer, vol. 7035, 2012, pp 250–259.
23. J. Scahill and J. Begley, "The Great SIM Heist," *The Intercept*, 19 Feb. 2015; https://firstlook.org/theintercept/2015/02/19/great-sim-heist.
24. S. Pell and C. Soghoian, "Your Secret Stingray's No Secret Anymore: The Vanishing Government Monopoly over Cell Phone Surveillance and its Impact on National Security and Consumer Privacy," *Harvard J. Law and Technology*, vol. 28, no. 1, 2014.



Lori Flynn is a software security researcher at CERT, in the Software Engineering Institute of Carnegie Mellon University. Contact her at lflynn@cert.org.



Will Klieber is a software security researcher at CERT, in the Software Engineering Institute of Carnegie Mellon University. Contact him at weklieber@cert.org.