Chad Spensky, Jeffrey Stewart, Arkady Yerukhimovich, Richard Shay, Ari Trachtenberg, Rick Housley, and Robert K. Cunningham

SoK: Privacy on Mobile Devices – It's Complicated

Abstract: Modern mobile devices place a wide variety of sensors and services within the personal space of their users. As a result, these devices are capable of transparently monitoring many sensitive aspects of these users' lives (e.g., location, health, or correspondences). Users typically trade access to this data for convenient applications and features, in many cases without a full appreciation of the nature and extent of the information that they are exposing to a variety of third parties. Nevertheless, studies show that users remain concerned about their privacy and vendors have similarly been increasing their utilization of privacy-preserving technologies in these devices. Still, despite significant efforts, these technologies continue to fail in fundamental ways, leaving users' private data exposed.

In this work, we survey the numerous components of mobile devices, giving particular attention to those that collect, process, or protect users' private data. Whereas the individual components have been generally well studied and understood, examining the entire mobile device ecosystem provides significant insights into its overwhelming complexity. The numerous components of this complex ecosystem are frequently built and controlled by different parties with varying interests and incentives. Moreover, most of these parties are unknown to the typical user. The technologies that are employed to protect the users' privacy typically only do so within a small slice of this ecosystem, abstracting away the greater complexity of the system. Our analysis suggests that this abstracted complexity is the major cause of many privacy-related vulnerabilities, and that a fundamentally new, holistic, approach to privacy is needed going forward. We thus highlight various existing technology gaps and propose several promising research directions for addressing and reducing this complexity.

Keywords: privacy-preserving technologies, mobile, Android, iOS

DOI 10.1515/popets-2016-0018
Received 2015-11-30; revised 2016-03-01; accepted 2016-03-02.

Chad Spensky: University of California, Santa Barbara, cspensky@cs.ucsb.edu
Jeffrey Stewart: MIT Lincoln Laboratory, jeffrey.stewart@ll.mit.edu

1 Introduction

The rapid proliferation of mobile devices has seen them become integral parts of many users' lives. Indeed, these devices provide their users with a variety of increasingly essential services (e.g., navigation, communication, and Internet connectivity), as well as useful functionality (e.g., entertainment and photography). To accommodate these services, modern mobile devices are equipped with various sensors, capable of collecting extremely rich information about their users and their surroundings. Users and developers alike have embraced these data-collection technologies with open arms, in exchange for the rich set of high-tech features that they enable. In fact, many of the *apps* offering these services are provided for "free," with an implicit exchange for access to portions of this collected data, which is typically used for advertising [1].

Unfortunately, there is also evidence that users do not understand the extent of information being collected about them or the personal details that can be inferred from this information. For example, a recent study on behavioral advertising showed that users did not understand how such advertising worked and found the information that the advertisers had about them to be "scary" and "creepy" [2]. Even when the users are aware of the data collection, one cannot expect them to fully realize the non-intuitive implications of sharing their data. Researchers have shown that the sensors on these devices can be used to covertly capture key presses (taps) on the phone [3, 4] or a nearby keyboard [5], leak the users' location [6–9], record speech [10], or infer the users' daily activities [11].

Predictably, having millions of users carrying these data-collection technologies presents numerous privacy

Arkady Yerukhimovich: MIT Lincoln Laboratory, arkady@ll.mit.edu
Richard Shay: MIT Lincoln Laboratory, richard.shay@ll.mit.edu
Ari Trachtenberg: Boston University, trachten@bu.edu
Rick Housley: Stevens Institute of Technology, rhousley@stevens.edu
Robert K. Cunningham: MIT Lincoln Laboratory, rkc@ll.mit.edu

concerns. The collected data may be accessible to a variety of parties, often without the explicit permission of the user. For example, companies that provide *trusted kernels* [12–14], which are relatively unknown to the user market, have the technical means to access most of the private user data that flows through their devices. Similarly, companies that advertise in mobile applications are able to learn a large amount of private information about the user (e.g., gender, age, or home address [15]), and some location-based applications have been shown to expose the users' fine-grained physical location not only to the application provider, but to the world [16, 17].

Recent high-profile data leaks have publicized compromising photographs [18], large numbers of credit cards [19], and National Security Agency (NSA) documents [20]. Leaks of financial, authentication, or medical data can be used to access costly services (such as medical operations [21]) and could result in damages that are extremely difficult to fix [22]. In this climate, it is not surprising that many users are concerned about their privacy [23] and have strong incentives to keep at least some of their data private. One survey, conducted in 2015, found that many Americans believed it to be "very important" to "be able to maintain privacy and confidentiality in [their] commonplace activities" [24], while another found that 57% of users avoided or uninstalled a mobile application because of privacy concerns [25].

In an attempt to address these concerns, numerous privacy-preserving technologies (PPTs) already exist at every level of the device stack (e.g., hardware, operating system, application), and some companies even highlight these privacy features as part of their marketing campaigns [26]. However, the complexity of the modern mobile-device environments, both hardware and software, and the piecemeal solutions offered by the individual components make security difficult to implement correctly and privacy difficult to achieve. In fact, the numerous components are produced and provisioned by disjoint entities with differing interests and incentives. In effect, complexity can be the enemy of *both* security *and* privacy, and the current on-device mobile ecosystem is especially complex.

In this work, we shed light on this complex system by surveying the numerous existing privacy-preserving technologies, the components that each relies upon, the parties involved, and the overall ecosystem they all cohabit. Our findings suggest that existing approaches have had marginal successes, but are likely going to continue to fall short of users' realistic privacy desires unless they are accompanied with fundamental changes to the ways that these devices are produced, provisioned, and regulated.

We scope our analysis to on-device technologies, which we consider the users' first line of defense. Still, we recognize that a lot of private data may also leave the device and enter a "cloud" infrastructure, introducing additional privacy concerns that we leave for future work. We also focus on identifying gaps in today's systems and, rather than providing concrete solutions, provide high-level and speculative suggestions for future research. Finally, whereas other studies have analyzed privacy on mobile devices (e.g., [3, 27–34]), we believe our work is the first to examine the entire ecosystem (i.e., hardware, software, human).

Mobile differentiators The problems that we have outlined underscore the unique challenges of the mobile device ecosystem with respect to privacy. These challenges arise from the very nature of the devices, which are *1)* extremely sensor-rich, *2)* continually proximate to their users, *3)* often accessed in public and through a limited interface, and *4)* produced in a rapidly-changing commercial environment. Neither laptops, with their limited sensors, nor desktops, with their limited user proximity, nor appliances, with their limited functionality and tightly controlled supply chain, provide the same level of deep-seated risks to the user's privacy.

Contributions Our work includes the following primary contributions, with the goal of painting a holistic picture of the overwhelming complexity in the mobile privacy space:

- We inventory critical privacy-related components on modern mobile devices, highlighting interactions that are likely unintended or unexpected.
- We identify the commercial entities currently responsible for deploying each technology.
- We itemize the various privacy-preserving technologies currently in use, calling attention to existing and historical flaws and gaps.
- We enumerate the types of private data that are currently accessible to the various parties.
- We describe several novel experiments that we performed to verify privacy-related claims.
- We highlight promising proposals that could help alleviate some of the existing problems.

Indeed, we conjecture that achieving privacy in the current ecosystem is unlikely without fundamentally novel technical and regulatory approaches.

2 Mobile Privacy Ecosystem

We begin our analysis by examining the components that comprise the mobile ecosystem (see Figure 1 for a brief summary). While mobile devices contain many of the same components as traditional computing platforms, and face many of the same privacy-related problems, there are fundamental differences at each layer that can impact user privacy.

The first differentiator between mobile and traditional computing platforms is the users' relationship with the devices, and the interfaces that they use to interact with them. More precisely, users typically carry their mobile devices, in a powered-on state, with them at all times, which is significantly different from the more stationary use-cases of both laptops and desktops. Subsequently, many user's have developed a unique psychological connection with their mobile device, in extreme cases developing an addiction [35]. Furthermore, while having a small form-factor is convenient for mobility, it also restricts potential user interactions. For example, while computers use separate input and output devices (e.g., keyboard, mouse, screen), these interfaces are frequently integrated and overlapping in mobile devices (e.g., touch screen), leading to various alternative interfaces (e.g., audio, vibration, motion). Moreover, this smaller, integrated, interface also restricts the amount of information that can be conveyed to or received from the user, which can lead to sub-optimal privacy implementations (e.g., omission of security indicators [36]).

The applications and operating systems that users interact with on mobile devices are also fundamentally different from traditional platforms. Historically, computers have provided relatively open environments, permitting developers essentially unfettered access to the system and similarly allowing users to install any application and modify their system in any way that they desired. However, mobile-device providers have adopted a more closed ecosystem, typically restricting the device to a single, pre-installed operating system, which is designed to only run applications written in a specific language (e.g., Java, Objective-C). These applications subsequently have very limited access to the system, using pre-defined application program interfaces (APIs), and are cryptographically attributed to an author. Mobile devices have also attracted a significant amount of attention from advertisers, due to the rich data available on these devices that can be used to perform targeted advertising [37]. This advertising-focused profit model has led to a significantly different economic model from that of traditional computers, as the users' private data has become the primary currency for most developers.

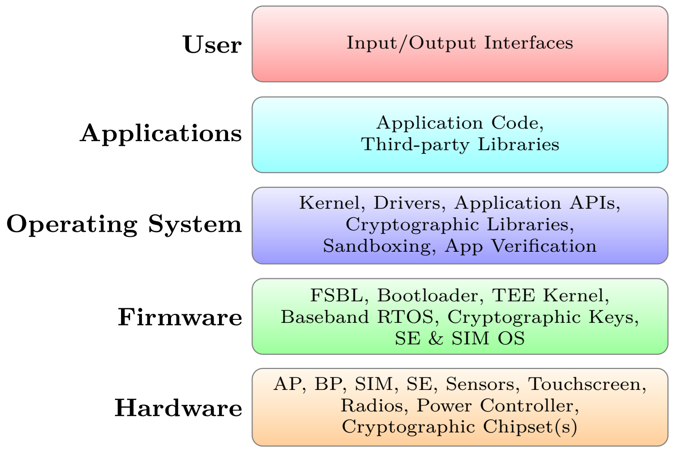

User
Input/Output Interfaces
Applications
Application Code, Third-party Libraries
Operating System
Kernel, Drivers, Application APIs, Cryptographic Libraries, Sandboxing, App Verification
Firmware
FSBL, Bootloader, TEE Kernel, Baseband RTOS, Cryptographic Keys, SE & SIM OS
Hardware
AP, BP, SIM, SE, Sensors, Touchscreen, Radios, Power Controller, Cryptographic Chipset(s)

Fig. 1. High-level abstraction of the components that constitute modern mobile devices.

Finally, mobile devices include a diverse medley of hardware components to provide the rich experience that users have come to expect. Due to the relative complexity of each component, they are typically produced by individual manufactures and then later combined into one unified device. Most of these hardware components have access to a lot of private information, which is not necessarily managed by the higher-level components on the device. Moreover, these hardware providers typically provide their own firmware and drivers to interact with their component, introducing even more complexity into the software environment.

2.1 Hardware Components

Mobile devices contain a plethora of individual hardware components that interact using various protocols. In some cases, the components can interact and expose sensitive user data in ways that were not intended. While the general hardware architecture of these systems has become extremely complex and varies wildly between devices, there are still some overarching similarities in both the included technologies and their interconnections. Figure 2 diagrams the typical hardware configuration found in modern smartphones.

Application Processor (AP) The application processors (APs) on mobile devices are very similar to central processing units (CPUs) found in traditional computers, with a few caveats. First, because of the stringent power requirements of mobile devices, many of the

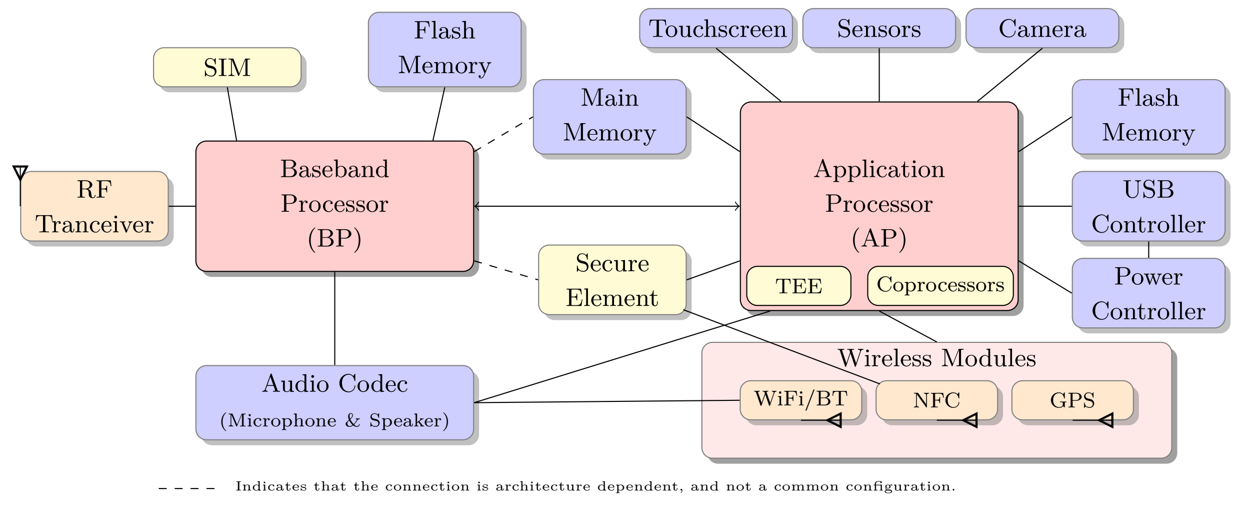


Fig. 2. Typical hardware configuration for a modern mobile device.

APs are designed with specialized low-power components and employ aggressive power-saving techniques (e.g., halting cores). Second, unlike legacy CPUs, many mobile devices utilize the system on a chip (SoC) model, wherein various sub-components (e.g., connectivity, media, location) are all included in a single silicon chip. Components within the SoC frequently share resources (e.g., memory), which can potentially leak private data. Finally, many of the newer APs include a trusted execution environment (TEE), which provides hardware-enforced isolation for "trusted" code, as well as the ability to arbitrate access to hardware components. These TEEs can be leveraged to enhance both security and privacy, but also present another layer of complexity.

Mobile Coprocessors In addition to the main AP, modern devices are equipped with additional coprocessors (e.g., Motorola's X8 [38] and Apple's M7 [39]) that assist the AP with various functions, each accompanied with their own firmware. More specifically, devices have recently been incorporating a natural language processor (NLP) for audio processing, and a contextual computing processor (CCP) for processing environmental information (e.g., motion, light). These coprocessors are typically used to offload computation from the AP and enable "always on" applications (i.e., the device can still be aware of it's surroundings while the AP and screen are in a power-saving state), which only wake the AP when a trigger event happens (e.g., saying "OK, Google," or handling the device). This always-on state could have serious privacy implications, as the users' devices are likely collecting data about them even when they believe the device to be "off."

Baseband Processor (BP) The baseband processors (BP) are typically multi-core processors that run a propriety real-time operating system (RTOS) as their firmware, which is provided by the BP vendor. Their sole purpose is to handle cellular communications, and they are thus typically isolated from the rest of the hardware on the device, with the exception of the microphone, speaker, and subscriber identity module (SIM). This permits the BP to remain performant when handling voice calls, regardless of the load on the other components. While the ideal implementation would enforce hardware isolation, it is typically cost effective to include the AP and BP on the same SoC, which could allow the BP to access main memory [40], and subsequently a lot of private user data, in addition to the cellular communications that it inherently controls.

Universal Integrated Circuit Cards (UICCs) A typical mobile device will contain at least two UICCs, a SIM card for authentication to a mobile carrier, and a secure element (SE) for handling user credentials on the device itself. Both of these units contain yet another completely self-contained processing unit in a security-hardened environment [41]. These components can be leveraged to protect user data and credentials by providing an isolated environment for cryptographic operations, which is typically leveraged for authentication and encryption. However, modern SIM cards can also perform other actions on the device (e.g., send short message service (SMS) messages, open a web browser, and send messages over the network [42, 43]), which could be used to violate the users' privacy.

Cryptographic Hardware In addition to SEs, some manufacturers have recently begun including dedicated cryptographic hardware for data encryption and authentication. For example, modern Apple devices have a dedicated module for encryption, using keys that are fused into the application processor during manufacturing to make them inaccessible from software [44]. These cryptographic modules can be utilized to protect the user's private data when the device is "locked."

Sensors Modern mobile devices are far more sensor-rich than traditional computers, and are equipped with a wide variety of sensing capabilities (e.g., movement, location, light). Additionally, numerous peripherals, or *wearables*, also enable access to pedometers, heart rate monitors, and even sensors to detect blood oxygen levels. These sensors are the source of much of the private data collected by mobile devices, as even access to a few of these sensors can provide incredible insights into the user's private life. For example, with just location and accelerometer data, Google is able to provide users with a complete log of their travels, including the mode of transportation [45].

2.2 Software Components

While the general software architecture of mobile devices is similar to that of traditional computers, there are a few key differences, which we briefly highlight.

Trusted Kernel Unlike historic computers, which only contained a single operating system, mobile devices frequently contain a separate, feature-rich, *trusted kernel* which is used inside the TEE. This trusted kernel is the most trusted code on the device. Its job is to create an isolated "non-secure," or "normal," world for the main operating system (e.g., Android, iOS) to execute within. The trusted kernel and the main operating system interact using very limited interfaces (e.g., shared memory regions), and the trusted kernel always maintains control of the non-secure world and has access to its entire memory space. Thus, the trusted kernel can undermine any data protections or privacy-preserving technology deployed by the OS provider.

Operating Systems The major differentiator between mobile OSs and their ancestors is primarily their closed nature. They are typically more restrictive in both policing which apps are allowed to be installed, and subsequently restricting the app's access once it is installed and executed. In order to restrict this access, mobile OSs employ sophisticated permissions models and application sandboxing techniques.

Mobile Applications In addition to the closed environment and advertising-focused economic model, the types of applications available on mobile devices are also significantly different from traditional computers. Specifically, mobile applications are very likely to make heavy use of their access to the various sensors. For example, numerous popular *social* applications leverage the users' precise location, and health-related applications collect intimate details about the users' personal lives. However, many of these applications are developed quickly with functionality as their primary focus and little thought given to user privacy, which can lead to this personal data being mishandled (e.g., users' intimate encounters appearing on search engines [46]).

Third-Party Libraries Many of the applications available for mobile devices make use of third-party libraries, such as targeted advertising libraries [33], for both functionality and profit. However, users are often unaware of the existence of these third-party libraries in the apps. What's worse, because there is currently no hard isolation between the components of an application, these libraries have the same access to data and sensors as their parent application, which is frequently leveraged to collect sensitive user data [15, 47].

2.3 Parties Involved

Numerous parties are typically involved in the deployment and provisioning of mobile devices, each responsible for different components. This significantly complicates privacy protection as the different parties may have varying incentives. Moreover, these parties often retain control of their components after deployment through over-the-air (OTA) updates. Understanding the parties involved, and their access to, and control of, the device is critical when analyzing the privacy assurances that are provided for mobile device data.

Chip Vendors Unlike the AP, which typically runs software that was provided by the device provider, both the BP and UICCs typically come with pre-installed software from their respective manufacturers. Because of these hardware components' privileged access in the device, a vulnerability or backdoor could provide access to private user data. In fact, Gemalto, a popular UICC manufacturer, recently made allegations that the United States' and Britain's intelligence agencies attempted to compromise their devices [48]. Furthermore, the area of *malicious hardware* has been gaining attention recently with recent publications showing just how practical these attacks are [49]. Detecting malicious hardware, if it exists, has been shown to be extremely difficult [50].

OS Provider The operating system (OS), and thus its provider, generally has access to a great deal of sensitive data. The OS is also responsible for ensuring adequate software-based segregation of applications and data sources. Most mobile OS vendors also employ remote administration capabilities, permitting them to provide updates post-deployment, as well as install and remove applications [51, 52]. This could enable them to change their protections or the amount of private data they collect, without informing users. More interestingly, many of the OS providers (e.g., Google, Apple) also have a strong incentive to collect private user data to leverage for advertising or their own services [53].

Trusted Kernel Manufacturers Recently, the desire to use the TEE has led to the advent of "trusted kernels," which come pre-installed on our mobile devices. Companies like *Trustonic* [54] and *TrustKernel* [13] provide this kernel to the device vendor, which then allow developers to install trusted applications within the TEE. Trustonic claims to have their kernel installed on more than 400 million devices, and permits over-the-air (OTA) updates of "Trusted Services"[14]. These parties have considerable power over mobile devices and an exploit in their firmware could be leveraged to completely compromise most security and privacy protections.

OEMs and ODMs The vendor that assembles the individual components into a mobile device for end users is called the Original Equipment Manufacturer (OEM) or Original Design Manufacturers (ODM) (e.g., Apple, Samsung, HTC, LG). These vendors design the hardware system architecture, the connections between the individual components, and often provide the first-stage boot loader (FSBL). In the case of the Android, OEMs typically modify the OS to incorporate specific hardware functionality, pre-install applications, add or remove features, and add services to ensure continued control over the device once it is deployed. Recently, OEM vendors have also started including their own interfaces and functionalities (e.g., Samsung's Knox[55]) in the TEEs of their devices. Even when the OEMs have the best of intentions, they are still capable of introducing severe vulnerabilities [56] as they are frequently increasing the complexity and attack surface of the OS.

Mobile Internet Service Providers (MISPs) Most modern devices rely heavily on Internet access provided by mobile Internet service providers (MISPs). In fact, many of the mobile devices on the market today are sold directly by MISPs. These MISPs inherently can observe all unencrypted traffic, and can trivially ascertain the rough physical location of devices using triangulation with their broadcast towers. The 3GPP, a partnership of communications interests, outlines the specifications for lawful interception of communications [57]. Many MISPs have far more access on devices than their role of providing Internet access would suggest. In many cases, they can remotely update the functionality of the SIM [58], which has access to numerous functions on the device enabling it to obtain global positioning system (GPS) coordinates, open a browser, and communicate with the BP and the AP.

Application Developers Mobile application developers inherently have the least-privileged access to the private data. However, most of the device's functionality to users is provided through these applications. Thus, exposing hardware sensors and other data sources to these applications is typically necessary. While most of these queries for data (e.g., GPS coordinates) will alert the user, it has been shown that most users will blindly accept these requests without much concern for how the data will be used [29].

Third-party Library Providers Instead of developing entire applications, some developers focus on creating libraries that can be included in other applications to provide specific functionality (e.g., cryptography, advertising, or graphics). Many developers use these libraries, exposing their data to the library provider. In particular, in exchange for monetary compensation, numerous applications include advertisement libraries. Researchers found that as many as 52.1% of applications utilize ad libraries [33]. Therefore, these ad-library providers can access a vast amount of user data spanning numerous applications and operating systems. Zang *et al.* found that 73% of Android apps shared personal information (e.g., name and email) and that 47% of iOS apps shared location data with third parties [59].

Mobile Device Management Many employers either provide mobile devices to their employees, or encourage them to *bring your own device* (BYOD), allowing employees to use their personal devices for business purposes. In both cases, the employer typically requires a mobile device management (MDM) solution to be installed, which provides the employer with a great deal of control over the device (e.g., install/blacklist applications, modify security settings, and location tracking). Employees storing personal data on MDM-enabled phones, and carrying them during off-business hours, presents some interesting privacy concerns that, to the best of our knowledge, have been left mostly unexplored.

2.4 Summary

The mobile-device ecosystem is a unique environment with numerous perils for user privacy. Unlike most desktop and laptop computers, mobile devices are filled with sensors that can collect a vast amount of sensitive data since the devices are typically always on and always with their user. Complicating things further, the hardware components of the mobile-device ecosystem, and their associated drivers, are provided by multiple vendors and potentially give these providers access to sensitive user data. Furthermore, software developers are allowed to distribute their own code to these devices in the form of apps with many of these monetizing the available user data through third-party advertisement libraries. The result is an ecosystem with multiple agents that have access to private user data and a clear incentive to gather, exploit, and share this private data.

3 Protecting Private Data

In this section, we review the numerous privacy-preserving technologies and techniques employed on modern mobile devices, across all levels of the stack. These protections are deployed in an effort to add security and privacy to the various components of the mobile ecosystem discussed in Section 2. In our presentation, we highlight the aspects of the technologies that make them particularly interesting on mobile devices. Additionally, we restrict our analysis to Google's Android and Apple's iOS platforms, which account for 97.8% of the smartphone market [60].

3.1 Hardware-based Mechanisms

Both Android and iOS leverage hardware-based features for protecting the privacy and security of sensitive data.

3.1.1 Trusted Execution Environment (TEE)

TEEs provide a hardware-enforced isolated execution environment for security-critical code, which can provide a safe haven for storing and processing sensitive data. The general concept behind TEEs is that this "trusted," or "secure," world, can be verified using techniques like secure or authenticated boot [61] and can be leveraged to maintain a root of trust on the device, even when the "normal," or "non-secure," world is compromised. The most popular of these, due to their market dominance, is ARM's TrustZone [62].

This functionality is available on all modern ARM cores and has only recently started to be leveraged to protect user data. For example, both Android [63] and iOS [44] are using this technology to protect fingerprint templates and Samsung has deployed Knox [55], which is used to provide segregation between the user's personal and work-related data on the device. However, vendors have only begun to scratch the surface, as this technology could also be leveraged to arbitrate access to the hardware-based sensors, potentially employing other privacy-preserving technologies to help thwart unintended disclosures. Similarly, it is possible to leverage the TEE to provide a secure *trusted path* (e.g., by dynamically reconfiguring the hardware permissions) for users to both input and digest sensitive information (e.g., passwords, private messages, pictures).

3.1.2 Data at Rest Encryption

Similarly, both Android and iOS leverage hardware-backed technologies to implement full-disk encryption, which protects all of the data on the device when it is "locked" or powered off. Specifically, both Android and iOS use the TEE to prevent the underlying encryption keys from off-device brute force attacks [44, 64]. Apple takes this protection a step farther by introducing a dedicated cryptographic hardware unit between the volatile system memory and non-volatile flash storage. Because the cryptographic hardware is separated from the AP, the TEE can provide it with encryption keys without exposing them to the OS. This protects the user's privacy by ensuring that even if an attacker can compromise the iOS kernel, they will be unable to steal the encryption keys needed to recover the data [44]. Android, on the other hand, implements full-disk encryption via the Linux kernel, based on dm-crypt [65], which stores the encryption key in kernel memory while in use.

The effectiveness of these techniques was recently demonstrated in a high-profile case, wherein the Federal Bureau of Investigation (FBI) was unable to recover data from an iOS device [66].

3.2 Software and User-based Mechanisms

We now describe the various tools used to control which applications may run on a mobile device, restrict the private data that these applications may access and collect, and protect the sensitive data once it is collected.

3.2.1 Application Vetting

One of the main selling points of modern mobile devices is their ability to run applications. In fact, the Android and iOS app markets both tout over 1.5 million apps, offered by a variety of third-party developers [67]. However, these developers can be untrustworthy or downright malicious [31, 68–72]. Therefore, both Google and Apple employ strict application vetting processes for their app stores, which check for violations of the developer policies and privacy-invasive behavior. While their approaches are quite similar, previous work suggests that vetted Android apps end up accessing less sensitive data than iOS apps [27].

Android Google offers an official app store, which it calls Google Play [73]. However, Google also allows users to purchase apps from a number of third-party app stores (e.g., Amazon [74]). In fact, the Google Play store is not available at all in China [75], forcing all Chinese users to obtain their apps from third-party stores.

Before being included in Google Play, an application must pass Google's vetting process, which ensures that the app does not violate the Google Play developer policies, many of which restrict the use of private user data [76]. Recently, Google's app-vetting process has started to include manual, human, checks [77] (similar to Apple's review process), in addition to their automated analysis.

However, this app-vetting process still suffers from a few deficiencies. First, the Android vetting process was historically only performed on applications that were submitted to Google Play. Thus, if the application was submitted to another app store or downloaded from an unknown source, the vetting was not performed; however, installing these applications still required explicit user consent. Nevertheless, Google recently introduced *Verify Apps Protection*, which automatically uploads and scans applications before installing from unknown sources [78]. Second, Android does not require the application code to be cryptographically signed before execution (i.e., developers can dynamically load program code after the vetting process). Therefore, a privacy-invasive application could pass the Google Play app vetting process, and then, when executed, download malicious code and run it. In fact, this technique has been abused by privacy-invasive malware in the wild [79].

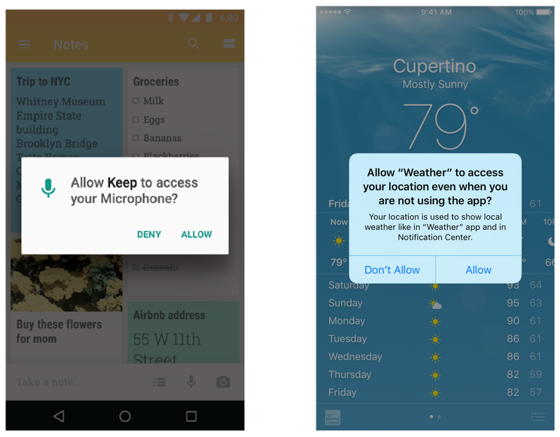
iOS Unlike Android, iOS only allows apps to be installed from the Apple App Store, which requires the app to have passed their vetting process. In addition to scanning for known malware and functionality checks, one of the major components in the Apple vetting process is to scan for applications that are using private APIs. While internally the low-level iOS Objective-C and C libraries contain many private methods, which are not documented in the official iOS API, iOS applications submitted to the App Store are required to only use the publicly documented APIs. Because these methods are called from within the public APIs, they cannot be completely removed from production. However, many of these methods are dangerous and have the ability to retrieve private information. For example, in our experimentation (detailed in Section 4.3.2), we found private API classes (i.e., *ACDServer*) that when initialized causes the main GUI on iOS 9.0 to crash. Additionally, previous work has found that many of these private APIs can be used to retrieve private data [72, 80–83].

3.2.2 Application Sandboxing

Both Android and iOS provide isolation mechanisms, called application sandboxes, which prevent applications from interfering with one another and from accessing resources that they do not possess permissions for. Both OSs reuse kernel-level security features that were previously designed for commodity operating systems to provide separation and a mechanism for enforcing mandatory access control (MAC).

Android Android's sandboxing capabilities center around Linux's user separation [84]. Android leverages this separation by assigning each installed application a unique user identifier (UID). The Linux kernel then enforces resource isolation to ensure that one application cannot interfere with another application (e.g., reading/writing their files, sending signals, debugging, etc.). Furthermore, Linux also provides Android with a mechanism to restrict an application's resource consumption (e.g., memory and CPU) and its access to the various device sensors. More recently, Android has also been employing SELinux's MAC framework to apply more fine-grained access control policies to applications [85].

iOS Unlike Android, far less is known about the mechanisms behind iOS's sandboxing implementation. The majority of what is known about Apple's sandbox implementation was learned through reverse engineering the implementation [86–88]. However, the underlying technology, TrustedBSD, is open source and detailed in Robert Watson's PhD Dissertation [89]. iOS's TrustedBSD MAC framework introduces additional checks in the kernel against the MAC profile's policy for various actions (e.g., reads and writes to a file, listing the files in a directory). These policies are provided by two kernel extensions: Apple Mobile File Integrity (AMFI), which implements the code signing and permission system; and *Sandbox.kext*, which implements the sandboxing components. Sandbox.kext implements sandboxing through the use of many built-in Scheme-like policies, which describe precisely what a process can and cannot do. There were 95 unique sandbox profiles found in iOS 8.0, and all third-party apps were assigned the "container" profile, which generally restricts the application to only access files within its own directory [88].


Allow **Keep** to access your Microphone?

Allow "Weather" to access your location even when you are not using the app?

Your location is used to show local weather like in "Weather" app and in Notification Center.

(a) Android (b) iOS

Fig. 3. User permission dialogs for Android [93] and iOS [94].

3.2.3 Permissions Models

While sandboxing applications prevents them from interfering with the device or stealing the users' data, there are plenty of legitimate reasons that apps may need to access sensitive resources. Both Android and iOS have run-time permissions models that require users to explicitly approve an application's request to access private data or potentially dangerous functionality.

Android The Android architecture divides permissions into two categories: *normal* and *dangerous* permissions. Dangerous permissions are needed for actions that appear to pose a risk to a user's privacy or the device's operations (e.g., location, phone operations, microphone), whereas normal permissions correspond to less risky actions (e.g., access the Internet, vibrate, set timezone), and may be revoked at any time. For dangerous permissions, user approval is required for all permission-controlled actions that it may undertake before the app will even be installed. Starting with Android 6.0, the user must also approve dangerous permissions the first time they are actually utilized; however, in the interest of usability, approval of one permission provides access to an entire group of permissions. For example, run-time approval of coarse-grain localization also covers later actions involving fine-grain localization. An example of one such permission dialog in Android is given in Figure 3a. This more recent permissions model resembles the permissions dialogs in iOS and aims to reduce over-permissioning [90], which arises when applications request a large number of permissions up front, in case they may be useful later. Previous research has shown that users often did not understand the install-time permissions dialogs and were often surprised by the data that these over-permissioned apps could access [29, 91, 92]. On the other hand, normal permissions don't pose such a risk, and are automatically granted at install time (i.e., the user is not prompted).

iOS The permission system in iOS uses two separate mechanisms to delegate access to the system, entitlements and permissions. Entitlements are rights set in an application's configuration and give the application a specific capability that it would otherwise not have (e.g., background processes, iCloud access) [95]. These entitlements are set in the app bundle that gets signed by Apple and therefore cannot be modified after the application is submitted to the App Store. Permissions, alternatively, are only approved at run-time, when the application actually attempts to access the restricted resource. When an application first attempts to access any restricted functionality or data (e.g., location, microphone, user contacts), iOS will prompt the user for approval (see Figure 3b), and the access will be granted only if the user agrees. The user's response to this prompt is saved, and any future attempt to access the same resource is subsequently approved. Unlike entitlements, which are approved by Apple when the application is submitted to the App Store, these permissions can be approved or revoked by the user at any time though iOS's menu system.

3.2.4 Data in Transit Encryption

Both Android and iOS provide APIs that allow applications to make secure network connections using SSL/TLS [96, 97]. These APIs ensure that applications do not leak private data to the MISP, a WiFi provider, or any malicious party that may be listening to external communications. In order to prevent the use of weak and broken cryptography, which may be vulnerable to cryptanalysis, Android disabled the use of weak cipher suites starting with Android 5.0 [98]. Apple has gone a step further, by mandating that applications on iOS 9.0 and later must use transport layer security (TLS) 1.2 with forward secrecy or an exception will be thrown, and the connection will not be made [99]. Additionally, in an attempt to reduce the number of unencrypted connections, Apple has begun forcing developers to specifically whitelist any domains that an application wants to make any unencrypted web requests to.

3.2.5 Privacy Policies

In addition to the technical tools for controlling access to private data, mobile devices also make use of regulatory mechanisms in an effort to protect users' privacy. Specifically, application developers often have privacy policies that govern the collection and use of private data by their apps. These policies are required by Apple and Google when accessing specific data on a device [100, 101], and are intended to inform users about how their information will be collected, stored, and used. Privacy policies can also sometimes be used for regulatory and legal action to enforce privacy protections; however, this is beyond the scope of this paper.

3.3 Summary

This section describes many of the tools currently deployed for protecting user privacy on mobile devices. An interesting observation is that there has been a significant focus on user-in-the-loop permissions to regulate access to this data (i.e., requiring explicit input from users). Moreover, regulating this access to private data is made even more important by the sheer number, and general untrustworthiness, of application developers. It is also worth noting that, for the most part, these privacy-preserving technologies exist in isolation, assuming the soundness of each other to ensure user privacy. For example, *1)* the OS assumes that the vetting process was successful, that the user will act appropriately when prompted for access to sensitive data (e.g., Figure 3), and that the application will not leak this data in unintended ways; *2)* the app assumes that the OS's sandboxing will protect it's data, and *3)* both the OS and app assume that the TEE and full-disk encryption will protect their stored data. While there have proposals that take a more system-wide approach for protecting user data [102, 103], we are unaware of any that are able to also account for the hardware components.

4 Remaining Privacy Leaks

Despite the best efforts of both the users and the technology providers, mobile devices continue to fail at protecting user data. Many of the issues stem from the complexity of the mobile ecosystem and have resulted in vulnerable implementations and fundamental design flaws. This section describes the remaining privacy leaks, within the various layers of the device, and the types of data leakages that result. These leakages are broadly summarized in Table 1, which demonstrates the significant exposure of vast amounts of private data, at all levels of abstraction, to a variety of parties.

4.1 Hardware Components: Baseband, SIM, and TEE

Many of the hardware components in mobile devices present unique challenges to the protection of user's private data. For instance, the Baseband Processor, SIM devices, and TEE are all generally considered black boxes; however, they can all access a substantial amount of user data, without much limitation.

The BP's role in the device's infrastructure allows it to observe all of the communications between the device and the MISP. This means that not only can it leak this private communication elsewhere, but that the BP can also be accessed and exploited through the same channel (e.g., via a nearby malicious cellular base station [126]). Xenakis *et al.* [30] showed that, once compromised, the BP can be used to access the device's GPS coordinates, calling number, international mobile subscriber identity (IMSI), and even the keys that are used to encrypt communication with the MISP via the SIM. Miller *et al.* [115] demonstrated a vulnerability in a BP's firmware that converted affected iPhones into a remote listening device, and a similar vulnerability was recently

Type of Data Available		Provider of Component											
		Hardware / Firmware				Software			Network		Malicious		
		AP	BP	SIM	TEE	OS	Apps	Libraries	MISP	MitM	Physical	App	Remote
Personal	Address Book	✓			[62]	✓	?	[59],[33]^A	[42, 43]	[56]^A	[104]^A[105]^i	[106][107]^A	[106][107]^A
	E-mail	✓			[62]	✓	?	[59]		[56]^A	[104]^A[105]^i	[106][107]^A	[106][107]^A
	Private Files	✓			[62]	✓	?			[56]^A	[104]^A[105]^i	[106][107]^A	[106][107]^A
	Unique Identifiers	✓	[30]	[42, 43]	[62]	✓	?	[59],[33]^A	[42, 43, 108]	[28, 108, 109][56]^A	[104]^A[105]^i	[106][107]^A[81, 83, 110, 111]^i	[106][107]^A
	Call/SMS (Logs)	✓	✓		[30]	✓		[33]^A	[57]	[56]^A	[104]^A[105]^i	[106][107]^A	[106][107]^A
	Call/SMS (Live)		✓		[30]				[57]	[112]^A	*N/A*	[106][107]^A[111]^i	[106][107]^A
Environment	Physical Location	✓	[113]	[42, 43]	[62]	✓	?	[59],[33]^A	[42, 43, 57]	[28, 109][56]^A	[104]^A[105]^i	[6–9, 106]	[16, 114]
	Microphone	✓	[113][115]^i		[62]	✓	?			[56]^A	[104]^A[105]^i	[10, 106][107]^A	[10, 106][107]^A
	Camera	✓	[113]		[62]	✓	?	[33]^A		[56]^A	[104]^A[105]^i	[106][107]^A[80, 81]^i	[106][107]^A
	Physical Sensors	✓			[62]	✓	?			[56]^A	*N/A*	[106][107]^A	[106][107]^A
Credentials	TEE	✓			✓						[104]^A	[116–118]	
	SIM & SE		[30]	✓	[30, 119]				✓		[120]	[30, 119]	[119]
	Apps	✓			[62]	✓		[59]		[121][56, 75, 122]^A	[104]^A[105]^i	[3, 4, 123][111]^i	[124, 125]

✓ - Access to the data is implied by the role of component ? - Access to the data is permitted after explicit user interaction
A - Android Only i - iOS Only

Table 1. Enumeration of which parties have been shown to have access to specific types of private data on modern mobile devices.

demonstrated against the Samsung's Galaxy S6 running Android [112]. In fact, the baseband on the iPhone has been exploited repeatedly to "unlock" iPhones, which were locked to a specific MISP [127]. Moreover, depending on the implementation, the BP could also be used to obtain data from user applications through the AP, or directly, using direct memory access (DMA) [113].

The great care given to the security of SIM devices makes them very difficult to analyze (e.g., most production SIMs do not permit the installation of third-party applications). Nevertheless, researchers have found vulnerabilities in SIM's protocols [128] and software [119], in addition to physical attacks [120], which permit access to the critical on-board data. These vulnerabilities could permit attackers to clone the device or decrypt network traffic, in addition to the other SIM capabilities (e.g., capturing GPS coordinates, sending messages, and displaying content on the users' screen [42, 43, 119]). In fact, a recent proof-of-concept malware was created that could obtain sensitive data from the SIM by querying it from the AP via the BP [30].

Finally, numerous vulnerabilities have been discovered in the implementation of the "secure world" code in TEEs (e.g., Huawei's HiSilicon [117], Qualcomm's MSM8974 [118] and Snapdragon [116, 129]). With access to code execution in this trusted world, an attacker can gain access to *all* of the data available on the device, including cryptographic keys, application data, biometric data, and sensor data. Thus, some research has focused on shrinking the trusted elements [130], so that they can be verified formally or by hand. Nevertheless, the popularity and trust placed in the TEE continues to grow together with the concern for privacy breaches.

4.1.1 Firmware/Drivers

Many of these hardware components contain their own firmware and are accompanied with drivers to enable communication with them. Most OS-providers blindly include these drivers, trusting them with privileged access to the system. However, drivers have long been exploited [131], and could pose a serious a risk to the user's privacy if they expose a vulnerability. In previously unpublished work, we analyzed the attack surface of near-field communication (NFC) interfaces on mobile devices by developing and exercising a physical-layer NFC fuzzing framework [132]. In this work, we examined the Nexus S and the Nexus 4, two flagship Android phones (running version 4.1.2 and 4.2.2 respectively). We conducted over 132,000 fuzz tests on the Nexus S, observing 856 crashes, and over 150,000 fuzz cases on the Nexus 4, observing only 7 crashes. Many of these crashes appeared in the drivers themselves, indicating that the significant difference in the number of observed crashes is likely due to the NFC chipset and their accompanying drivers, as the software versions were essentially identical. This experiment further emphasizes the compounded complexity that occurs in mobile devices, with their various third-party hardware dependencies.

4.2 Sensors

Unlike general purpose computers, almost all mobile devices contain a GPS sensor for detecting physical location. In some cases, this location information could be used to infer a significant amount about the user's activities (e.g., illegal, embarrassing, or "secret" locations) or personal characteristics (e.g., at a club that caters to particular social interests). While numerous protections exist to protect this valuable data (e.g., requiring explicit user-granted permission), numerous side-channels have been shown, which still leak accurate location data. For example, proximity-based dating apps share the user's location with the provider to find potential nearby dates, and these providers have been shown to leak the user's location (with varying granularities) to anyone with an app-related connection to the user [16, 114]. Furthermore, local WiFi network signal strength [6] or availability [133], accelerometers [7], and even ambient sound and light [9] have also been shown to leak location information.

Users also frequently enter private credentials into their devices through user-interfaces such as the touchscreen, camera, or microphone. These user-interfaces, restricted by the size of the device and commonly accessed in public venues, can leak access to data over significantly more channels than their less-mobile desktop and laptop counterparts. Beyond a bystanders' ability to view the screen or overhear conversations, it has been shown that the keyboards on touch screens are inherently susceptible to being observed [124], even when direct line-of-sight is not available [125]. Similarly, data from the device's accelerometer and/or microphone has been leveraged to infer user key presses [4] and tap locations [3, 123] on the touch screen. These methods could be used to covertly intercept users' passwords, financial information, and private communications from a background app with access to the sensors. Further, Michalevsky *et al.* [10] recently demonstrated that the gyroscope, which is accessible to applications and websites in both Android and iOS, could be utilized to identify speaker information and even parse speech.

4.3 Operating System

Due to the OS's visibility, and its position as the first line of defense against malicious applications, it is unsurprising that almost every aspect of the OS has subsequently been targeted and exploited to obtain private data. We highlight the most notable examples below.

4.3.1 Bloatware and Updates

One of the major distinctions between Apple and Android devices relates to their control over the device: Apple has significantly more control over many of the components with which iOS interacts (including hardware), whereas Android is more commonly modified and distributed by various third-party vendors before it reaches the user. For example, Apple restricts the applications that are bundled with the device, whereas Android devices typically come with a variety of such applications pre-installed, often with unwanted and unknown (to the user) privacy-invasive features [34, 134, 135]. Unlike desktops and laptops, which may also come with pre-installed bloatware, users on mobile devices often have no means to remove it, resulting in a much longer sustained privacy invasion. These Android modifications have led to vulnerabilities [136], and some vendors have abused this privilege by adding "special features" (e.g., remote access) to the devices [137, 138]. Most notably, Samsung was recently accused of modifying the version of Android running on their devices to implement remote-control functionality from the BP [139].

Similarly, because of the distributed Android ecosystem, system updates can take a much longer time to apply to all devices than for iOS, where Apple can push updates unilaterally. The result of these delayed updates was shown in a recent study of the Android ecosystem, which found that 87.7% of devices were vulnerable to at least one critical vulnerability [140], partially due to the slow updates. This is in stark contrast to general purpose computing, where patches for critical vulnerabilities are often deployed as quickly as possible.

4.3.2 Permissions and Sandboxing

As described previously, both Android and iOS have sandboxes and permissions systems to restrict third-party application developers from accessing the private sensor data available on the mobile platforms.

Android The Android permissions model is designed to allow users of Android devices to control what data may be collected by the applications installed on their devices [141]. In principal, this should allow the user to be aware of all sensitive data accessed by any application installed on their device. However, as our experiments detailed below indicate, this is not always the case. Often times, unintended side-channel attacks permit the recovery of more information than was originally intended (e.g., taps [3, 4], keypresses [5], position [6, 7, 9, 10], current activity [11]). For instance, Michalevsky *et al.* recently demonstrated the ability to recover a phone's location from the seemingly innocent ability of an application to measure the phone's aggregate power use [8].

iOS As mentioned in Section 3.2.1, Apple has specifically forbidden the use of private APIs by applications, because they may leak private user data. However, there has been a large body of work on bypassing the application vetting process in order to allow private API usage. For example, Han *et al.* [80] were able to get several privacy-invasive apps (e.g., secret filming and screen capturing) through Apple's vetting by utilizing private API calls in dynamically loaded and obfuscated libraries. Other applications have successfully evaded the vetting process by implementing hidden remote vulnerabilities, which, post-vetting, permitted remote agents to divert execution into a previously "dead" code branch that would dynamically load the needed libraries to make private API calls [81].

To prevent these attacks, Bucicoiu *et al.* used static binary rewriting to modify an iOS application's code to include a reference monitor to restrict the use of private APIs [82]. Similarly, Deng *et al.* developed a system, using static analysis, dynamic analysis, and forced execution, to detect private API use and found 146 approved that were using private APIs [83]. Furthermore, another system, that used static analysis and simple string de-obfuscation techniques, was able to identify 256 approved apps using these privacy-invasive APIs [142].

A design flaw in Apple's enterprise app model has been shown to permit the publication of completely unvetted applications to the public [111]. Because these applications are signed with an enterprise certificate, they are not vetted by Apple and can utilize private APIs without restriction. Recent research analyzing these publicly distributed enterprise apps found that 80% of the analyzed apps do, in fact, utilize private APIs, primarily to obtain private user information [111]. Unsurprisingly, iOS malware has also utilized this technique to infect un-jailbroken devices [72, 143, 144].

Experiment In order to analyze the existing permissions models for sensitive data leakage, we conducted an experiment to determine the amount of private data that is currently available to applications with *no* explicit permissions [145].

On Android (version 5.1.1), we used Java's reflection to traverse the class hierarchy in the runtime environment, allowing each of the fields and methods for all of the available classes to be accessed and analyzed for private data. While complications arise from the need to supply the correct arguments for object constructors and other methods, these can be mitigated by recursively creating the needed arguments and utilizing the results of previous method calls. This technique identified numerous unique identifiers that could be used to fingerprint the device (e.g., directory structures and account authenticators), discern the user's interests (e.g., installed applications and associated metadata), obtain the user's personal information (e.g., device wallpaper), and learn the fine-grained per application network throughput, which can be exploited for a number of side-channel attacks (e.g., decrypting voice over IP (VoIP) [146–148], website fingerprinting [149]).

This experiment was similarly conducted on iOS (versions 8.1.3 and 9.0); however, the effectiveness was hindered by a number of complications, such as Objective-C's lack of memory safety, type safety, and complete reflection support. However, our unprivileged test application was still able to obtain a few noteworthy results: modifying the device's volume without user interaction; causing the main GUI, `SpringBoard`, to crash; causing other processes, such as `geod` (the location provider service daemon) to crash; and causing the OS kernel to crash.

Over-Permissioning One of the simplest methods of collecting private data from a user is to simply request access to it upon installation. Once the user agrees to the permissions, future updates of the app can then exercise those permissions. Various studies have approximated that a significant amount of Android applications (one-third of 940 [90], 44% of 10,000 [150]), in fact, request permissions that they do not use. Moreover, this trend of over-permissioning appears to be increasing in applications [151] and the use of these permissions has been similarly growing in advertisement libraries [152].

Data collected by an over-permissioned application can typically be exfiltrated to the Internet, since such access is provided by default on both Android and iOS. To combat this risk, several systems attempt to identify user data that is sent off device by an application. PiOS uses static analysis on iOS applications to detect code flows which first access user data and then utilize that data in network API calls [153]. TaintDroid took this analysis one step further by modifying Android to conduct system-wide dynamic taint tracking to detect network transmission of user data [103].

4.3.3 Full Disk Encryption (FDE)

On Android, the OS is also responsible for managing the full disk encryption, which is based on the Linux kernel's *dm-crypt*. Previous work has shown that once an application can read the Linux kernel's memory, it is possible to steal these encryption keys [154], revealing the sensitive data. Similarly, Android is also vulnerable to memory dumps, as shown by Müller *et al.* [104], who obtained the encryption keys via a cold boot attack.

4.4 Application

In addition to the various OS-level protections, application developers also have a strong incentive (i.e., reputation) to protect the private user data that they are legitimately collecting, and thus employ various techniques to implement these protections.

4.4.1 Data in Transit Encryption

Unless network encryption is properly used, any private data sent over the network could be trivially intercepted (e.g., by a Man-in-the-Middle (MitM) or wireless provider). As such, there have been a large number of studies investigating applications' use of encryption in transit on both iOS and Android [75, 121, 122, 155–163], revealing a wide range of flaws that leaked user data.

More precisely, *mallodroid* [164], an open-source static analysis tool for Android, found that 1,074 out of 13,500 analyzed applications were improperly using cryptographic certificates (41 of the 1,074 were manually verified) [122]. Similarly, *SMV-Hunter* [165], another open-source analysis tool for Android, found 1,453 potentially vulnerable apps using static analysis (726 of which were confirmed using dynamic analysis) to be vulnerable to MitM attacks [155]. Moreover, numerous applications (87.9% of the analyzed Android apps [156] and 65.3% of analyzed iOS apps [157]) also have poorly implemented cryptographic constructs (e.g., using ECB mode for encryption or hard-coded encryption keys), which could expose private user data through relatively simple attacks.

Experiment While Google and Apple have since taken some steps to reduce the misuse of SSL/TLS, as described in Section 3, they do not appear to have completely solved the problem. Our analysis of 50 recent banking applications (with both Android and iOS versions) found 4 apps on iOS and 2 apps on Android incorrectly validating SSL/TLS certificates. In this work, we were using a Samsung Galaxy S6 and an iPhone 6s, both with the latest versions of the applications at the time for Android 5.1.1 and iOS 8.1.3. One noteworthy anecdote from our analysis suggests things are improving however; one application had originally appeared vulnerable when analyzed statically, but before we could analyze the application dynamically it required an update that properly utilized TLS.

4.4.2 Privacy Policies

As mentioned previously, privacy policies are used to govern an applications' use of private user data and to inform users of how their data is being used. In principal, these policies should enable users to make informed decisions about what applications to install, based on their privacy preferences. However, their efficacy is solely dependent on the users' ability to understand the information contained within these policies. To examine their effectiveness, we analyzed the privacy policies of the top 50 paid and top 50 free apps for Android and the top 100 paid and top 100 free apps for iOS (56 unique Android policies and 126 unique iOS policies) for their readability using the *Flesch Kincaid Grade Leve* (FKGL) [166]. Our results indicate that privacy policies on both Android and iOS, on average, require a college reading level, and that very few of the policies, on either platform, were understandable at a high-school reading level. Thus, while these policies are supposed to serve as the primary tool for conveying privacy-related information to users, they are likely incomprehensible for a large fraction of the population and, for the most part, ineffective.

4.5 A Data-Centric View

It is also interesting to take a data-centric view of this privacy picture (i.e., the rows in Table 1). That is, for specific classes of data, to consider how this data can be accessed and at what level of the platform.

The first class of data that we examine are unique identifiers. These can be used to track users across services, applications, or devices potentially allowing a malicious party to learn a person's history through such connections. Note that such identifiers are accessible at all levels of the stack. The different components of the hardware layer are given this data as part of their operations. As our experiments showed, many types of identifiers can be accessed without any permissions, thus giving access to any applications or third-party libraries that may be installed on the device. Additionally, several works have shown how this private data can be accessed by an eavesdropper or the MISP [108].

Another interesting class of data to look at is location data, which is clearly very sensitive, as access to it allows parties to locate and track the user of the mobile device. Unfortunately, a number of prior works [16, 42, 43, 56, 59, 104, 105, 113, 114] have shown that the picture here isn't much better. Location data is largely available via various attacks and leakages for most devices across the different levels of the stack, revealing this private information to unintended parties. For instance, advertising libraries were able to obtain a user's location by appearing in an application with location permissions [33], MISPs can obtain a user's location through normal operations [57], as can the BP and SIM. It was also recently shown that even a passive observer is able to locate a particular user by intercepting unauthenticated communication with the MISP [28, 109].

4.6 Summary

As this section shows, despite the many privacy protections employed, privacy leaks continue to persist on mobile devices. This problem is exacerbated by the various channels, both explicit (i.e., user permissions) and implicit (i.e., a side-channel), that this data is accessible through. Subsequently, because the various parts of the mobile stack are built and controlled by different providers, this private data is accessible to a wide variety of parties. Furthermore, the abundance of sensors on mobile devices leads to the collection and possible leakage of much more private data than would be available in other computing environments. By examining the privacy gaps (both component-focused and data-focused) at each level of the stack, we are able to appreciate the overwhelming complexity that exists within these devices. To this end, while protecting user data that is explicitly exposed may be a tractable problem, a solution that is also able to protect data from the more implicit interactions will require significantly more effort.

5 A More Private Future

As can be easily gleaned from our survey, the current mobile-device ecosystem is extremely complex, with many components that are controlled by a variety of parties. This complexity leads to a number of privacy leaks, which are not entirely addressed by existing privacy-preserving technology. In this section, we suggest a few possible research directions, aimed at reducing this complexity to improve mobile privacy. We note that our recommendations are very high-level and are meant to suggest broad research directions, rather than prescribe specific solutions or guaranteed fixes.

Reducing Trust Relationships One way to reduce the complexity of the mobile device ecosystem is to reduce the number of trust relationships that must be established between the users and the various components of the ecosystem. One particularly promising way to do this is through the use of advanced cryptographic techniques such as homomorphic encryption [167], secure multi-party computation [168, 169], and functional encryption [170]. There are already surveys presenting the current state of these technologies (e.g., [171, 172]). These techniques would allow users to directly control who can access their private data and what they can do with it, without relying on any other privacy-preserving component to enforce these policies. While these techniques are all currently in the early research stages, we believe that in the near future some of these technologies will make a major impact in this space.

Other ways to reduce these trust relationships include tools such as information flow control [102, 173], secure programming languages [174], and formal program verification [175]. Systems enforcing information-flow control (e.g., [176]) can ensure that sensitive data is only accessed by certain prescribed components, and in appropriate ways, without requiring trust in these components. Similarly, secure programming languages and program verification can be used to ensure that low-level firmware and applications are built in ways that do not expose significant privacy vulnerabilities.

Guiding Users Toward Privacy In addition to reducing the number of trust relationships, there is a need to help end users make good privacy-related decisions to protect their own private data. For example, both Android and iOS currently prompt users when an app would like to access specific data. However, it has been shown that users generally do not understand the implications of these decisions [29, 92]. Several suggestions for helping users understand these decisions have been proposed in the literature. Wang *et al.* [177] proposed adding purposes to permissions requests to inform users what their data will be used for, while Agarwal *et al.* [178] used crowdsourcing to help users determine which permissions were really necessary for a given application. Another suggested approach aims to help users by splitting permissions needed for the core application and any third-party libraries that these applications may include [179–182], thus reducing any confusion as to which parties will be accessing the data.

Mechanism Design for Privacy Finally, with all of the parties and complexity in the mobile ecosystem, we could use a self-regulating approach to preserving privacy. One such approach could be to leverage game theory, specifically the area of *mechanism design* to incentivize privacy-preserving behavior from selfish agents. This is similar (in principle) to how Bitcoin [183] encourages cooperation from its untrusted users. One suggestion to move toward such economic incentives would be to impose financial punishments on culpable parties in case of a private-data leak, possibly also leveraging tools such as data provenance to identify and punish all parties involved in collecting the leaked data. Alternative approaches include setting up auctions for permissions to collect user data, and allowing users to charge more for access to more-sensitive data. Of course a highly nontrivial part of this approach would be to identify parties to enforce the penalties and rules of such mechanisms. One suggestion may be to leverage the Bitcoin crypto currency to enforce such punishments without relying on any centralized trusted party as in the work on fair exchange [184].

6 Conclusions

In this work, we holistically analyzed the current mobile ecosystem and the impact that it is having on users privacy. Our analysis uncovered the overall complexity of these devices and the numerous parties associated with the components at each layer in the stack (e.g., hardware, software, third-parties), each with their own, sometimes conflicting, perspectives on user privacy. While many of these parties have taken significant efforts to introduce protections for their individual components, these protections have not generally withstood attack from the research community. In fact, data leakage has been found in every layer of the stack, compromising a wide variety of components, and in many cases leveraging a vulnerability in one component to leak data from another, otherwise protected, component. We thus codified how personal, environmental, and credentialing data are often *de facto* accessible to hardware, software, and network providers that service the mobile phones, meaning that a privacy failure of *any* of these providers can lead to significant leakage of private data.

In effect, our survey and experiments have demonstrated that, despite great efforts by companies with exceptional resources, existing privacy-preserving technologies have fallen short of ensuring user privacy in an age were unprecedented amounts of data about our most intimate moments has become available to our omnipresent mobile devices. We have argued that the main cause for this failure has been the enormous complexity of the mobile device ecosystem with its multitude of involved components and parties. In our view, the current approach, wherein each component attempts to manage its own privacy protections, will not ensure user privacy going forward. Instead, we believe a fundamentally different approach is needed, which expressly addresses the complexity and unique capabilities of this mobile ecosystem. To this end, we have proposed several potential research directions, with the hope that they will eventually lead to a more private tomorrow.

Acknowledgments

We would like to thank John Launchbury at DARPA for making this entire project possible and our collaborators at the RAND Corporation for their invaluable input.

This material is based on work was sponsored by DARPA under Air Force Contract #FA8721-05-C-0002 and in part on work supported by the National Science Foundation under Grant No. CNS-1012910. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DARPA or the National Science Foundation.

Approved for public release: distribution unlimited.

References

[1] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo, "Don't kill my ads!: balancing privacy in an ad-supported mobile application market," in *MobiSys 2012*.

[2] B. Ur, P. G. Leon, L. F. Cranor, R. Shay, and Y. Wang, "Smart, useful, scary, creepy: perceptions of online behavioral advertising," in *SOUPS 2012*.

[3] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *WiSec 2012*.

[4] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." in *HotSec 2011*.

[5] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *CCS 2011*.

[6] J. Krumm and E. Horvitz, "LOCADIO: Inferring motion and location from Wi-Fi signal strengths," in *MobiQuitous 2004*.

[7] J. Han, E. Owusu, L. T. Nguyen, A. Perrig, and J. Zhang, "Accomplice: Location inference using accelerometers on smartphones," in *COMSNETS 2012*.

[8] Y. Michalevsky, G. Nakibly, A. Schulman, and D. Boneh, "PowerSpy: Location tracking using mobile device power analysis," in *USENIX Sec. Symp.* USENIX Association, 2015.

[9] M. Azizyan, I. Constandache, and R. Roy Choudhury, "Surroundsense: mobile phone localization via ambience fingerprinting," in *MobiCom 2009*.

[10] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing speech from gyroscope signals," in *USENIX Sec. Symp.* USENIX Association, 2014.

[11] L. Sun, D. Zhang, B. Li, B. Guo, and S. Li, "Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations," in *Ubiquitous intelligence and computing*. Springer, 2010, pp. 548–562.

[12] Qualcomm, "Haven sec. platform," https://www.qualcomm.com/products/snapdragon/security.

[13] TrustKernel Team, Shanghai Pingbo Info Tech Co., Ltd., "Trustkernel," https://www.trustkernel.com/.

[14] J. Bennett, "Devices with trustonic tee," https://www.trustonic.com/news-events/blog/devices-trustonic-tee, 08 2015.

[15] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter, "Free for all! assessing user data exposure to advertising libraries on android," in *NDSS 2016*.

[16] I. Polakis, G. Argyros, T. Petsios, S. Sivakorn, and A. D. Keromytis, "Where's wally?: Precise user discovery attacks in location proximity services," in *CCS 2015*.

[17] C. Patsakis, A. Zigomitros, and A. Solanas, "Analysis of privacy and security exposure in mobile dating applications," in *MSPN 2015*.

[18] R. McCormick, "Hack leaks hundreds of nude celebrity photos," http://www.theverge.com/2014/9/1/6092089/nude-celebrity-hack, Sep. 2014.

[19] B. Krebs, "The target breach, by the numbers," *Krebs on Security*, vol. 6, 2014.

[20] "Newly disclosed N.S.A. files detail partnerships with AT&T and Verizon," *The New York Times*, 2015.

[21] K. M. Sullivan, "But doctor, I still have both feet! Remedial problems faced by victims of medical identity theft," *American Journal of Law & Medicine*, vol. 35, no. 4, 2009.

[22] C. Apgar, G. Apple, L. Ayers, M. Berntsen, R. Busch, J. Childress, E. Curtis, N. Davis, M. Dawson, B. Hjort *et al.*, "Mitigating medical identity theft," *Journal of American Health Information Management Association*, vol. 79, no. 7, p. 63, 2008.

[23] C. J. Hoofnagle and J. M. Urban, "Alan Westin's privacy homo economicus," *Wake Forest Law Review*, 2014.

[24] M. Madden and L. Rainie, "Americans' attitudes about privacy, sec. and surveillance," http://www.pewinternet.org/2015/05/20/americans-attitudes-about-privacy-security-and-surveillance/, May 2015.

[25] J. L. Boyles, A. Smith, and M. Madden, "Apps and privacy: More than half of app users have uninstalled or decided to not install an app due to concerns about their personal information," http://www.pewinternet.org/2012/09/05/main-findings-7/, Sep. 2015.

[26] Apple, "We've given you tools to manage your privacy," http://www.apple.com/privacy/manage-your-privacy/, Retrieved Nov. 2015.

[27] J. Han, Q. Yan, D. Gao, J. Zhou, and R. Deng, "Comparing mobile privacy protection through cross-platform applications," in *NDSS 2013*.

[28] A. Shaik, R. Borgaonkar, N. Asokan, V. Niemi, and J.-P. Seifert, "Practical attacks against privacy and availability in 4G/LTE mobile communication systems," in *NDSS 2016*.

[29] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *SOUPS 2012*.

[30] C. Xenakis and C. Ntantogian, "Attacking the baseband modem of mobile phones to breach the users' privacy and network security," in *CyCon 2015*.

[31] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proc. 2012 Symp. on Sec. and Privacy*. IEEE, 2012.

[32] L. Li, A. Bartel, T. F. D. A. Bissyande, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, "IccTA: Detecting inter-component privacy leaks in android apps," in *ICSE 2015*.

[33] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *WiSec 2012*.

[34] P. McDaniel, "Bloatware comes to the smartphone," *IEEE Sec. & Privacy 2012*.

[35] W. Park, "Mobile phone addiction," *Mobile Communications*, 2005.

[36] C. Amrutkar, P. Traynor, and P. C. van Oorschot, "An empirical evaluation of security indicators in mobile web browsers," *Transactions on Mobile Comp.*, vol. 14, no. 5, 2015.

[37] I. Paul, "Google's new, highly targeted app ads react to how you use android apps," http://www.pcworld.com/article/2147001/google-starts-using-your-android-app-behavior-to-deliver-highly-targeted-app-ads.html, 2015.

[38] Motorola, "X8 mobile computing system," http://www.motorola.com/us/X8-Mobile-Computing-System/x8-mobile-computing-system.html.

[39] Apple, “iPhone 6s Technology,” http://www.apple.com/iphone-6s/technology/.

[40] R. Krten, “Google android – IPC at the lowest levels,” http://www.embedded.com/print/4083262, June 2008.

[41] W. Rankl and W. Effing, “Smart card security,” *Smart Card Handbook, 4th Edition*, pp. 667–734, 2010.

[42] K. Koscher and E. Butler, “simhacks,” http://simhacks.github.io/.

[43] ETSI, “Smart Cards; Card Application Toolket (Release 13),” March 2015.

[44] Apple, “iOS Security: iOS 9.0 or later,” https://www.apple.com/business/docs/iOS_Security_Guide.pdf, 2015.

[45] Google, “Google history,” https://history.google.com/.

[46] C. Matyszczyk, “TMI? Some fitbit users’ sex stats on Google search,” http://www.cnet.com/news/tmi-some-fitbit-users-sex-stats-on-google-search/, Retrieved Nov. 2015.

[47] S. Son, D. Kim, and V. Shmatikov, “What mobile ads know about mobile users,” *NDSS 2016*.

[48] Gemalto, “Gemalto presents the findings of its investigations into the alleged hacking of sim card encryption keys by britain’s government communications headquarters and the U.S. National Security Agency,” 2 2015.

[49] J. Zhang, F. Yuan, and Q. Xu, “DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans,” in *CCS 2014*.

[50] S. Wei and M. Potkonjak, “The undetectable and unprovable hardware trojan horse,” in *DAC 2013*.

[51] T. Bray, “Exercising our remote application removal feature,” http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html, June 2010.

[52] C. Beaumont, “Apple’s Jobs confirms iPhone ‘kill switch’,” http://www.telegraph.co.uk/technology/3358134/Apples-Jobs-confirms-iPhone-kill-switch.html, Aug. 2008.

[53] Google, “What data does Google collect?” https://privacy.google.com/data-we-collect.html.

[54] Trustonic, “Trustonic,” https://www.trustonic.com/.

[55] Samsung, “Samsung knox,” http://www.samsungknox.com/, Nov. 2015.

[56] R. Welton, “Remote code execution as system user on samsung phones,” https://www.nowsecure.com/blog/2015/06/16/remote-code-execution-as-system-user-on-samsung-phones/, June 2015.

[57] ETSI, “3G security; Lawful Interception; Stage 2 (3GPP TS 43.033 version 12.0.0 Release 12),” Oct. 2014.

[58] S. Gemplus, Oberthur, “Over-the-air (OTA) technology,” ftp://www.3gpp.org/tsg_sa/WG3_Security/TSGS3_30_Povoa/Docs/PDF/S3-030534.pdf, Oct. 2010.

[59] J. Zang, K. Dummit, J. Graves, P. Lisker, and L. Sweeney, “Who knows what about me? A survey of behind the scenes personal data sharing to third parties by mobile apps,” http://techscience.org/a/2015103001/, 2015.

[60] V. Woods and R. van der Meulen, “Gartner says emerging markets drove worldwide smartphone sales to 15.5 percent growth in third quarter of 2015,” http://www.gartner.com/newsroom/id/3169417, 2015.

[61] J.-E. Ekberg, K. Kostiainen, and N. Asokan, “Trusted execution environments on mobile devices,” in *CCS 2013*.

[62] ARM, “ARM Sec. Technology: Building a Secure System using TrustZone Technology,” 2009.

[63] H. Lockheimer, “Hi, I’m Hiroshi Lockheimer, here at Google with the team that build Nexus 5X & 6P...Ask Us Anything!” https://www.reddit.com/r/IAmA/comments/3mzrl9/hi_im_hiroshi_lockheimer_here_at_google_with_the/cvjj167, Oct. 2015.

[64] “Full disk encryption,” https://source.android.com/security/encryption/, 2015.

[65] M. Broz, “dm-crypt: Linux kernel device-mapper crypto target,” https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt, 2015.

[66] J. Bonneau, “A technical perspective on the apple iphone case,” https://www.eff.org/deeplinks/2016/02/technical-perspective-apple-iphone-case, 2 2016.

[67] Statistica, “Number of apps available in leading app stores as of july 2015,” http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, July 2015.

[68] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” in *SPSM 2011*.

[69] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, “Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets.” in *NDSS 2012*.

[70] “Mobile Phone Spy Software,” http://www.mobistealth.com/mobile-phone-spy-software, 2015.

[71] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, “ANDRUBIS-1,000,000 apps later: A view on current android malware behaviors,” in *BADGERS 2014*.

[72] Claud Xiao, “YiSpecter: First iOS Malware That Attacks Non-jailbroken Apple iOS Devices by Abusing Private APIs,” http://researchcenter.paloaltonetworks.com/2015/10/yispecter, 2015.

[73] Google, “Google play,” https://play.google.com/.

[74] Amazon, “Amazon appstore,” http://www.amazon.com/mobile-apps/b?node=2350149011.

[75] F. Cai, H. Chen, Y. Wu, and Y. Zhang, “Appcracker: Widespread vulnerabilities in user and session authentication in mobile apps,” in *MoST 2015*.

[76] Google, “Google play developer program policies,” https://play.google.com/about/developer-content-policy.html, 2015.

[77] Kim, Eunice, “Creating better user experiences on google play,” http://android-developers.blogspot.ro/2015/03/creating-better-user-experiences-on.html, 2015.

[78] Google, “Android security 2014 year in review,” https://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_2014_Report_Final.pdf, 2014.

[79] Cluley, Graham, “The Hacking Team Android malware app that waltzed past Google Play’s security checks,” https://heatsoftware.com/security-blog/10368/the-hacking-team-android-malware-app-that-waltzed-past-\google-plays-security-checks/, 2015.

[80] J. Han, S. M. Kywe, Q. Yan, F. Bao, R. Deng, D. Gao, Y. Li, and J. Zhou, “Launching Generic Attacks on iOS with Approved Third-Party Appl.” in *Applied Cryptography and Network Sec.* Springer, 2013, pp. 272–289.

[81] T. Wang, K. Lu, L. Lu, S. Chung, and W. Lee, “Jekyll on ios: When benign apps become evil,” in *USENIX Sec. Symp.* USENIX Association, 2013.

[82] M. Bucicoiu, L. Davi, R. Deaconescu, and A.-R. Sadeghi, “XiOS: Extended application sandboxing on iOS,” in *ASIACCS 2015*.

[83] Z. Deng, B. Saltaformaggio, X. Zhang, and D. Xu, “iRiS: Vetting private API abuse in iOS applications,” in *CCS 2015*.

[84] “System and kernel security,” https://source.android.com/devices/tech/security/overview/kernel-security.html, 2015.

[85] “Security-Enhanced Linux in Android,” https://source.android.com/security/selinux/, 2015.

[86] D. Blazakis, “The apple sandbox,” in *Black Hat DC*, 2011.

[87] D. A. Dai Zovi, “Apple iOS 4 security evaluation,” https://www.trailofbits.com/resources/ios4_security_evaluation_paper.pdf.

[88] S. Esser, “iOS8 Containers, Sandboxes and Entitlements,” in *Ruxcon*, 2014.

[89] R. N. M. Watson, “New approaches to operating system security extensibility,” University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-818, Apr. 2012.

[90] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in *CCS 2011*.

[91] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, “A conundrum of permissions: installing applications on an android smartphone,” in *Proc. Financial Cryptography and Data Sec.* Springer, 2012, pp. 68–79.

[92] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, “Expectation and purpose: understanding users’ mental models of mobile app privacy through crowdsourcing,” in *Ubicomp 2012*.

[93] “Android 6.0 marshmallow,” https://www.android.com/versions/marshmallow-6-0/, 2015.

[94] Apple Inc., “About privacy and Location Services for iOS 8 and iOS 9,” https://support.apple.com/en-us/HT203033, 2015.

[95] ——, “Entitlement Key Reference,” https://developer.apple.com/library/ios/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/AboutEntitlements.html, 2014.

[96] “Using networking securely,” https://developer.apple.com/library/prerelease/ios/documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/SecureNetworking/SecureNetworking.html, 2015.

[97] “Security with https and ssl,” http://developer.android.com/training/articles/security-ssl.html, 2015.

[98] “Security enhancements in android 5.0,” https://source.android.com/security/enhancements/enhancements50.html, 2015.

[99] “What’s new in iOS: iOS 9.0,” https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS9.html#//apple_ref/doc/uid/TP40016198-DontLinkElementID_13/, 2015.

[100] “App Store Review Guidelines,” https://developer.apple.com/app-store/review/guidelines/, 2015.

[101] “Google Play Developer Distribution Agreement,” https://play.google.com/about/developer-distribution-agreement.html, 2015.

[102] L. Jia, J. Aljuraidan, E. Fragkaki, L. Bauer, M. Stroucken, K. Fukushima, S. Kiyomoto, and Y. Miyake, “Run-time enforcement of information-flow properties on android,” in *Computer Security – ESORICS 2013*. Springer, 2013, pp. 775–792.

[103] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones,” *TOCS 2014*.

[104] T. Müller and M. Spreitzenbarth, “Frost,” in *Applied Cryptography and Network Security*. Springer, 2013, pp. 373–388.

[105] J. Zdziarski, “Identifying back doors, attack points, and surveillance mechanisms in iOS devices,” *Digital Investigation*, vol. 11, no. 1, pp. 3–19, 2014.

[106] U. D. of Justice, “Pakistani man indicted for selling stealthgenie spyware app,” https://www.fbi.gov/washingtondc/press-releases/2014/pakistani-man-indicted-for-selling-stealthgenie-spyware-app, Sep. 2014.

[107] P. Coogan, “Android rats branch out with dendroid,” http://www.symantec.com/connect/blogs/android-rats-branch-out-dendroid, March 2014.

[108] T. Chen, I. Ullah, M. A. Kaafar, and R. Boreli, “Information leakage through mobile analytics services,” in *HotMobile 2014*.

[109] D. F. Kune, J. Koelndorfer, N. Hopper, and Y. Kim, “Location leaks on the GSM air interface,” in *NDSS 2012*, 2012.

[110] D. Richardson, “XcodeGhost iOS malware: The list of affected apps and what you should do,” http://blog.lookout.com/blog/2015/09/21/xcodeghost-apps/, Sep. 2015.

[111] M. Zheng, H. Xue, Y. Zhang, T. Wei, and J. C. Lui, “Enpublic apps: Security threats using iOS enterprise and developer certificates,” in *ASIACCS 2015*.

[112] P. Paganini, “Snooping Samsung S6 calls with bogus base stations,” http://securityaffairs.co/wordpress/41923/hacking/snooping-samsung-s6.html, Nov. 2015.

[113] R.-P. Weinmann, “Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks.” in *WOOT*. USENIX Association, 2012.

[114] G. Qin, C. Patsakis, and M. Bouroche, “Playing hide and seek with mobile dating applications,” in *ICT Sys. Sec. and Privacy Protection*. Springer, 2014, pp. 185–196.

[115] C. Miller, D. Blazakis, D. DaiZovi, S. Esser, V. Iozzo, and R.-P. Weinmann, *iOS Hacker’s Handbook*. John Wiley & Sons, 2012.

[116] D. Rosenberg, “Reflections on trusting trustzone,” in *Blackhat*, 2014.

[117] D. Shen, “Attacking your trusted core: Exploiting trustzone on android,” in *Blackhat*, 2015.

[118] laginimaineb, “Full trustzone exploit for msm8974,” http://bits-please.blogspot.com/2015/08/full-trustzone-exploit-for-msm8974.html, 2015.

[119] K. Nohl, “Rooting Sim Cards,” in *BlackHat Briefings*, Las Vegas NV, July 2013.

[120] J. R. Rao, P. Rohatgi, H. Scherzer, and S. Tinguely, “Partitioning attacks: or how to rapidly clone some GSM cards,” in *Symp. on Sec. and Privacy*. IEEE, 2002.

[121] J. Hubbard, K. Weimer, and Y. Chen, “A study of SSL proxy attacks on Android and iOS mobile applications,” in *CCNC 2014*.

[122] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, “Why eve and mallory love android: An analysis of android SSL (in)security,” in *CCS*

2012.

[123] S. Narain, A. Sanatinia, and G. Noubir, "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning," in *WiSec 2014*.

[124] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "iSpy: automatic reconstruction of typed input from compromising reflections," in *CCS 2011*.

[125] Y. Xu, J. Heinly, A. M. White, F. Monrose, and J.-M. Frahm, "Seeing double: Reconstructing obscured typed input from repeated compromising reflections," in *CCS 2013*.

[126] R.-P. Weinmann, "New challenges in baseband exploitation: The hexagon architecture," *CODEGATE 2014*.

[127] iPhone DevTeam, "Evolution of the iPhone Baseband and Unlocks," http://old.sebug.net/paper/Meeting-Documents/hitbsecconf2012ams/D1T2%20-%20MuscleNerd%20-%20Evolution%20of%20iPhone%20Baseband%20and%20Unlocks.pdf, May 2012.

[128] O. Dunkelman, N. Keller, and A. Shamir, "A practical-time attack on the A5/3 cryptosystem used in third generation GSM telephony," *IACR Cryptology ePrint Archive*, vol. 2010, p. 13, 2010.

[129] S. Beaupre, "Trustnone," http://theroot.ninja/disclosures/TRUSTNONE_1.0-11282015.pdf, 2015.

[130] Y. Gilad, A. Herzberg, and A. Trachtenberg, "Securing smartphones: A μtcb approach," *Pervasive Comp.*, vol. 13, no. 4, pp. 72–79, 2014.

[131] H. Chen, Y. Mao, X. Wang, D. Zhou, N. Zeldovich, and M. F. Kaashoek, "Linux kernel vulnerabilities: State-of-the-art defenses and open problems," in *APSys 2011*.

[132] C. Spensky and H. Hu, "LL-SmartCard," https://github.com/mit-ll/LL-Smartcard.

[133] G. Wilkinson, "Digital terrestrial tracking: The future of surveillance," 2014.

[134] Eckhart, Trevor, "Carrier IQ part 2," https://www.youtube.com/watch?v=T17XQI_AYNo, 2011.

[135] N. Lee, "Smartphones and privacy," in *Facebook Nation*. Springer, 2014, pp. 71–84.

[136] "CVE-2014-8346." Available from MITRE, CVE-ID CVE-2014-8346., Oct. 10 2014.

[137] G. Data, "G data mobile malware report: Threat report q2/2015," https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/G_DATA_MobileMWR_Q2_2015_EN.pdf, 2015.

[138] D. Gilbert, "Amazon selling 40 dollar android tablets that come with pre-installed malware," http://www.ibtimes.com/amazon-selling-40-android-tablets-come-pre-installed-malware-2181424, 2015.

[139] P. Kocialkowski, "Samsung galaxy back-door," http://redmine.replicant.us/projects/replicant/wiki/SamsungGalaxyBackdoor, Februrary 2014.

[140] D. R. Thomas, A. R. Beresford, and A. Rice, "Security metrics for the android ecosystem," in *SPSM 2015*.

[141] "System permissions," http://developer.android.com/guide/topics/security/permissions.html, 2015.

[142] SourceDNA, "iOS apps caught using private APIs," https://sourcedna.com/blog/20151018/ios-apps-using-private-apis.html, 2015.

[143] Z. Chen, A. Mettler, P. Gilbert, and Y. Kang, "iBackDoor: High-Risk Code Hits iOS Apps," https://www.fireeye.com/blog/threat-research/2015/11/ibackdoor_high-risk.html, 2015.

[144] C. Xiao, "WIRELURKER: A new era in iOS and OS X malware," https://www.paloaltonetworks.com/content/dam/paloaltonetworks-com/en_US/assets/pdf/reports/Unit_42/unit42-wirelurker.pdf, 2015.

[145] J. Stewart, A. Trachtenberg, and A. Yerukhimovich, "Look ma, no permissions! accessing private data on android," In Submission, 2016.

[146] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose, "Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks," in *Proc. Symp. on Sec. and Privacy*. IEEE, 2011.

[147] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson, "Language identification of encrypted VoIP traffic: Alejandra y roberto or alice and bob?" in *USENIX Sec. Symp.* USENIX Association, 2007.

[148] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Uncovering spoken phrases in encrypted voice over IP conversations," *TISSEC 2010*.

[149] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *Symp. on Sec. and Privacy*. IEEE, 2012.

[150] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) permissions used by android apps," in *MSR 2013*.

[151] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," in *ACSAC 2012*.

[152] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of android ad library permissions," in *MoST 2013*.

[153] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: Detecting privacy leaks in iOS applications," in *NDSS 2011*.

[154] C. Carmony, "dm_dump," https://github.com/c1fe/dm_dump/, 2014.

[155] D. Sounthiraraj, J. Sahs, G. Greenwood, Z. Lin, and L. Khan, "SMV-Hunter: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in android apps," in *NDSS 2014*.

[156] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *CCS 2013*, New York, NY, USA.

[157] Y. Li, Y. Zhang, J. Li, and D. Gu, "iCryptoTracer: Dynamic analysis on misuse of cryptography functions in iOS Applications," in *Network and System Sec.* Springer, 2014, pp. 349–362.

[158] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking SSL development in an appified world," in *CCS 2013*.

[159] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating SSL certificates in non-browser software," in *CCS 2012*.

[160] L. Onwuzurike and E. De Cristofaro, "Danger is my middle name: experimenting with SSL vulnerabilities in android apps," in *WiSec 2015*.

[161] N. Vallina-Rodriguez, J. Amann, C. Kreibich, N. Weaver, and V. Paxson, "A tangled mass: The android root certificate stores," in *CoNEXT 2014*.

[162] B. Reaves, N. Scaife, A. Bates, P. Traynor, and K. R. Butler, "Mo(bile) money, mo(bile) problems: analysis of branchless banking applications in the developing world," in *USENIX Sec. Symp.* USENIX Association, 2015.

[163] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl, "To pin or not to pin? Helping app developers bullet proof their TLS connections," in *USENIX Sec. Symp.* USENIX Association, 2015.

[164] "Mallodroid," https://github.com/sfahl/mallodroid, 2015.

[165] "Smv-hunter," https://github.com/utds3lab/SMVHunter, 2015.

[166] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," Naval Technical Training Command, Tech. Rep., 1975.

[167] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC 2009*.

[168] A. C. Yao, "Protocols for secure computations (extended abstract)," in *FOCS 1982*.

[169] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC 1988*.

[170] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *TCC 2011*.

[171] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and performance of programmable secure computation," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1039, 2015.

[172] S. Yakoubov, V. Gadepally, N. Schear, E. Shen, and A. Yerukhimovich, "A survey of cryptographic approaches to securing big-data analytics in the cloud," in *HPEC 2014*.

[173] J. Yang, K. Yessenov, and A. Solar-Lezama, "A language for automatically enforcing privacy policies," in *SIGPLAN Notices*, vol. 47, no. 1. ACM, 2012, pp. 85–96.

[174] A. Ruef and C. Rohlf, "Programming language theoretic sec. in the real world: A mirage or the future?" in *Cyber Warfare*. Springer, 2015, pp. 307–321.

[175] H. K. Harton, M. Sitaraman, and J. Krone, "Formal program verification," *Wiley Encyclopedia of Comp. Science and Engineering*, 2008.

[176] A. A. de Amorim, N. Collins, A. DeHon, D. Demange, C. Hritcu, D. Pichardie, B. C. Pierce, R. Pollack, and A. Tolmach, "A verified information-flow architecture," in *POPL 2014*.

[177] H. Wang, J. Hong, and Y. Guo, "Using text mining to infer the purpose of permission use in mobile apps," in *UbiComp 2015*.

[178] Y. Agarwal and M. Hall, "ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing," in *MobiSys 2013*.

[179] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, in *ASIACCS 2012*.

[180] S. Shekhar, M. Dietz, and D. S. Wallach, "Adsplit: Separating smartphone advertising from applications," in *USENIX Sec. Symp.* USENIX Association, 2012.

[181] X. Zhang, A. Ahlawat, and W. Du, "Aframe: isolating advertisements from mobile applications in android," in *ACSAC 2013*.

[182] H. Kawabata, T. Isohara, K. Takemori, A. Kubota, J.-i. Kani, H. Agematsu, and M. Nishigaki, "Sanadbox: Sandboxing third party advertising libraries in a mobile application," in *ICC 2013*.

[183] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, 2011.

[184] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *CCS 2015*.