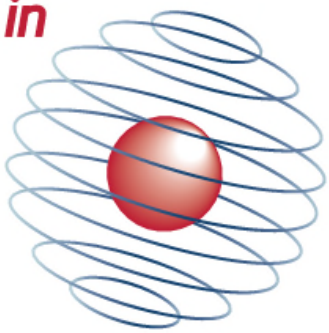
CENTRE for DOCTORAL TRAINING in
CYBER SECURITY



CDT Technical Paper

03/14

Android Apps and Privacy Risks: What Attackers can Learn by Sniffng Mobile Device Traffic

Vincent Taylor

Android Apps and Privacy Risks: What Attackers can Learn by Sniffing Mobile Device Traffic

Vincent F. Taylor, Jason R.C. Nurse, and Duncan Hodges

Cyber Security Centre, Department of Computer Science,
University of Oxford, United Kingdom.
{vincent.taylor, jason.nurse, duncan.hodges}@cs.ox.ac.uk

Abstract. Recent years have witnessed significant growth in the mobile device landscape as smartphones and tablet computers have become more affordable and more feature-rich. Users commonly extend the functionality built into these mobile devices by installing add-on applications, called apps. Many popular apps adopt a client-server architecture and communicate with Internet-based services to provide users with a rich and dynamic experience. Worryingly, some of these apps need to access sensitive data such as phone-book entries, appointments, messages, or a user's geographic location, but precisely how apps use and transmit sensitive data over wireless networks has not been widely studied. We examine the traffic sent from 35 popular Android apps spread over 6 categories to explore what an attacker with a promiscuous wireless receiver could learn about a target. We discovered that the majority of the apps that were tested had a detrimental impact on privacy by sending sensitive data without encrypting it. We also discovered that in some cases, improper application design rendered SSL encryption useless at preventing privacy leaks. We discuss ways in which an attacker can use both active and passive attacks to identify and track a user or invade their privacy. Finally, we suggest and discuss several possible solutions to mitigate the privacy risks that were identified.

Keywords: Android, privacy, mobile app, sniffing, smartphone

1 Introduction

As smartphones and tablet computers become ubiquitous and more powerful, users are relying on them to store more sensitive data. This sensitive data is not only a user's personal information and their correspondence with contacts, but also the cache of data (such as geographic location) from the sensors that are built in to these mobile devices. Mobile device applications, hereafter called 'apps', are an easy way to add additional functionality to a device. On Google's Android platform, when a user begins the process of installing an app, they are presented with an ultimatum in the form of a dialogue box which asks them whether they are willing to grant the particular set of permissions that the app has requested. The user is forced to accept all the permissions or abandon the installation altogether. Apple's rival mobile operating system, iOS, handles app permissions in a different way, but an analysis of that architecture is outside of the scope of this paper. While many apps will have legitimate reasons to access sensitive information, there is no way to know, a priori, the extent to which the application actually uses this information and whether the application is sending this sensitive information to third parties for secondary purposes such as advertising or even more malevolent goals such as harvesting data for targeted attacks.

Many apps are provided free of cost with the developers generating revenue by embedding advertisements, hereafter called 'ads', into their apps. Typically, ads are delivered to a user by *ad providers* who supply developers with libraries which are used to simplify *ad delivery*. To make ads more targeted to the end-user, ad providers will usually try to gather as much information about the user as possible to try to gain insight into their interests and purchasing habits. Ad providers have a vested interest in delivering highly suitable ads because these usually lead to higher click-through rates and thus more revenue for the ad provider. For this reason, when fetching an ad, ad libraries usually send any available information about the user or device to the ad server to help it to provide a more relevant ad. The data sent by ad libraries, as well as other data transmitted by the app, contribute to the privacy concerns surrounding apps on mobile devices.

We are concerned with two main actors as it relates to privacy leakage from mobile devices running the Android Operating System (OS). The first actor is the ad network that sends potentially private information to help better target ads to end-users. More sophisticated ad networks may try to track users across apps (using identifiers unique to a device, for example IMEI number) to gain a better understanding of the types of apps that a particular user is interested in. The second actor is the attacker that sits somewhere on the communication path between end device and server and tries to eavesdrop on this data with the intention of exploiting it for their own gain. For example, an attacker in a coffee shop could passively sniff the traffic coming from smartphones and tablet computers connected to the coffee shop's public wireless network. That attacker could potentially harvest a significant amount of personal data from those patrons that visited the coffee shop and connected to the wireless network.

For this research, we are concerned with what data is being gathered and transmitted by apps as well as whether private information is being transmitted securely to its destination. We evaluate 35 popular applications from 6 categories and analyse their network traffic as an attacker would. We examine the extent to which encryption is used by apps and identify ways that an attacker can still harvest information about a victim even when encryption strategies are used.

The rest of this paper is organised as follows: Section II presents a brief background on Android and reflects on several papers discussing its privacy concerns. Section III describes the threat model that we assume throughout our experiment. In Section IV, we describe the testing methodology that was used to collect data from apps. Section V presents the results of our experiment and explains their relevance to the privacy problem. Section VI discusses our findings and highlights issues and solutions to the problems identified. Section VII concludes the paper and suggests areas for future work.

2 Background

Android is a Linux-based operating system that is designed primarily for smartphones and tablet computers. It is the most popular mobile operating system and is shipped on more mobile devices than Windows, iOS and Mac OS combined [8]. Each Android application runs in its own sandbox and can only access privileged system resources through the Android API if the relevant permission was granted at install time. An application declares what permissions it requires in its manifest file and users are presented with this list of permissions when they choose to install

the application. The application installation only continues when the user grants the application all of its requested permissions.

App developers routinely use ad libraries to add advertising functionality to their applications [3]. On the Android OS, these libraries run in the same process space as the application and thus have the same capability and access to privileged resources as the application [14]. There are a number of sources on an Android device that an application or ad library can pull sensitive data from. Sensitive resources and data include, but are not limited to: location, phone number, telephony device ID (IMEI, MEID or ESN), unique device identifier, call state, contacts, user account information, microphone, browser history and bookmarks, logs, MAC addresses, SMS messages and calendar. We define a privacy leak as an opportunity for an attacker to capture the transmission of any data that was pulled from a sensitive source on the device.

Most of the sources of sensitive data outlined above require explicit permission before they can be accessed. Naive users are arguably more vulnerable to privacy leaks because they are unable or unwilling to make an informed choice when deciding whether an application is requesting more permissions than it reasonably needs. Mylonas et al. [13] conducted a survey which suggested that users are complacent about app security since they trust the official app repositories and thus disregard security during the installation of an app. In another study, Felt et al. found that participants displayed low attention and comprehension rates in relation to Android permissions when installing apps [7]. They found that only 17% of participants paid attention to the permissions being requested by applications at install time. This is worrying because non-permission-hungry applications are already leaking sensitive data from smartphones to the Internet [6], and a vast majority of users not checking what permissions an application is asking for only compounds the problem.

Beresford et al. propose a modified version of the Android operating system, called MockDroid, which allows a user to control an application's access to a resource at runtime [5]. MockDroid works by allowing a user to revoke an app's access to a particular resource by intercepting the request and giving a fake response or making it appear that the resource is unavailable. The authors argue that apps are already built to tolerate resource failure (such as lack of a GPS signal) so MockDroid will allow most apps to continue to function normally, or with reduced functionality, while preserving privacy. The MockDroid approach is very powerful and allows users to discriminate when allowing different applications access to the device's resources. The average uninformed user may not necessarily care to tinker with privacy settings for each application so MockDroid may not be an effective solution for fully combating the problem of privacy leaks in Android mobile devices.

Hornyack et al. also examine privacy controls for Android smartphones that limit the exfiltration of private data [10]. They substitute real responses with "shadow data" when an untrusted application tries to access data that the user has marked as being private. The authors also go a step further and offer a way of blocking network transmissions that contain data that the user only allowed to be used on the device. The efficacy of this system was tested using an automated process that captured visual differences in the output from the applications with and without the privacy controls being applied. The authors found that they could be able to limit permissions of applications with no adverse effects in 66% (33 of 50) of the applications that they tested. This work provides a valuable contribution

but also does not cater to the average uninformed user that does not want to spend the time to configure what application is allowed to access what data.

Enck et al. propose a system called TaintDroid which leverages information flow tracking to monitor the movement of sensitive data on an Android device [6]. TaintDroid works by labelling sensitive data and automatically tracking the flow of this data as it propagates through medium such as program variables and files. TaintDroid is able to identify when this sensitive data leaves the system and keeps track of what application was responsible, the type of data and the data's destination. The authors' hope is that the real-time feedback provided to users gives greater insight into what apps are doing so that it may be easier to identify privacy-eroding applications. This approach is useful and offers insights into the behaviours of apps but also fails to address the needs of the naive user that does not know or care about analysing the output from the system.

Yang et al. propose a unique strategy for combating the leakage of sensitive data [16]. They argue that some applications have legitimate reasons to transmit private data and that what needs to be checked is whether the transmission was user intended. The authors identify transmissions that are user intended by analysing GUI manipulations that ultimately lead up to data transmission. They argue that unintended data transmission could be those transmissions that happen in an irregular fashion and irrelevant to the function that the user is performing with the device. The authors make a strong assumption that a transmission that was not intended by the user is more likely to be a privacy leakage. This assumption fails to account for applications that have legitimate reasons to send data in the background, such as an app that syncs with a server. This approach also gives less scrutiny to data transmissions that were as a result of user interaction. One could envisage a malicious app defeating this system by only sending private data when a user is directly interacting with the app to make the transmission look legitimate.

A number of authors propose static analysis techniques for detecting privacy leaks in Android applications [9] [11] [12]. Kim et al. propose a tool, called ScanDal, which they argue can detect every possible privacy leak in an application. ScanDal determines if there is any flow of sensitive data from a source on the device to a sink. Using ScanDal, they were able to identify a number of privacy leaks in popular applications. Mann and Starostin [12] and Gibler et al. [9] propose similar frameworks and tools for static analysis and demonstrate the efficacy of using the static analysis approach to identifying privacy leaks in applications. The static approach scales well and may be a useful tool for an app repository curation team. Some static detection systems are unable to properly handle implicit flows within an application and so additional research needs to be done if these tools are to gain significant traction.

While static analysis is useful to identify apps that may leak private data, we need to remember that this leaking of private data is sometimes required for a particular app to carry out its task; such as a map application that sends GPS data to obtain a map of the surrounding area. Static analysis frameworks may be able to identify a potential leak but would not be able to differentiate between the legitimate passing of data to a server and the undesired passing of data to a server.

From the literature, we can see that a number of authors have proposed interesting methods of solving the problem of privacy leakage in Android applications. These methods are a useful start to solving the problem but no one method is a panacea, and each has its limitations. Regardless of what steps are taken to reduce the problem of privacy leakage in general, there is still the problem of the well-

known man-in-the-middle (MITM), an attacker that resides on the communication link between victim and server. Our work is novel because we analyse the traffic sent from mobile apps as an attacker would, to gain a greater understanding of the data that is at risk.

3 Threat Model

As mentioned in Section 2, we consider a privacy leak to be an opportunity for an attacker to capture traffic containing data that originated from a sensitive source on the mobile device. For this reason, we are not concerned with whether the application doing the transmission is malicious or whether the user has consented to the transmission. Our focus is analysing traffic once it leaves the device, regardless of intent, to determine the extent to which an attacker somewhere on the communication path from client to server is able to obtain sensitive information that could be used to identify the device or the device's user.

The attacker we are concerned about ranges from one which is able to passively sniff traffic or perform a MITM attack on the local network segment, to one who can capture arbitrary traffic from anywhere on the communication link from client to server. In the first instance, our analysis is not concerned with an attacker directly interfering with a victim's device such as exploiting vulnerabilities in the Android OS, the Android SDK, or the Dalvik Virtual Machine that runs the Android applications. However, in Section 6 we discuss what active attacks an attacker could deploy after having passively collected information about a target.

4 Methodology

As our objective was to gain the best insight into the traffic that passes from the average user's Android mobile device, we adopted an experimental approach to identify what an attacker would be able to learn if they were on the same network as the user. For our experiment, we used an Android smartphone, but the results of our analysis can be generalised to any mobile device that has the Android OS installed. We chose to analyse apps that could be downloaded free of charge for two reasons:

1. Free applications are more widely installed than paid applications [1].
2. Free applications tend to be ad-supported (and use ad libraries) so these may leak more private data than paid applications.

For our selection, we chose apps from the official Google Play Store, a virtual marketplace containing a large collection of Android applications, and the default location for Android devices to source apps. In the Google Play Store, apps are placed in different general categories and there is a feature that allows a user to browse the most popular apps for each category as well as the most popular apps in the store. The top 5 apps from 5 categories as well as 10 randomly chosen apps from the top 100 were selected to get good coverage of the apps that might be installed on the average user's device. The categories chosen were Communication, Games, Productivity, Health & Fitness and Shopping. We chose each category for the following reasons:

– Communication: These apps are widely installed by the userbase and may handle private conversations between one or more parties.

- Games: These apps are widely installed by the userbase.
- Health & Fitness: These apps potentially handle sensitive medical information that would be valuable to advertisers or insurance companies.
- Productivity: These apps may be used for work collaboration and thus may handle potentially sensitive corporate data.
- Shopping: These apps may handle monetary transactions and would provide valuable insights to advertisers.
- Random: Randomly choosing apps allowed us to get a sample of popular applications outside of the categories that were selected.

The list of apps selected for analysis using our sampling methodology is shown in Table 1. We also show the number of downloads of each app as supplied by the Google Play Store. Only apps that performed network communication were relevant to the analysis, but the initial selection of apps did not have to be altered because they all performed network communication. Throughout this paper, we preserve the capitalisation and format of app names as they are presented by the developers in the Google Play Store.

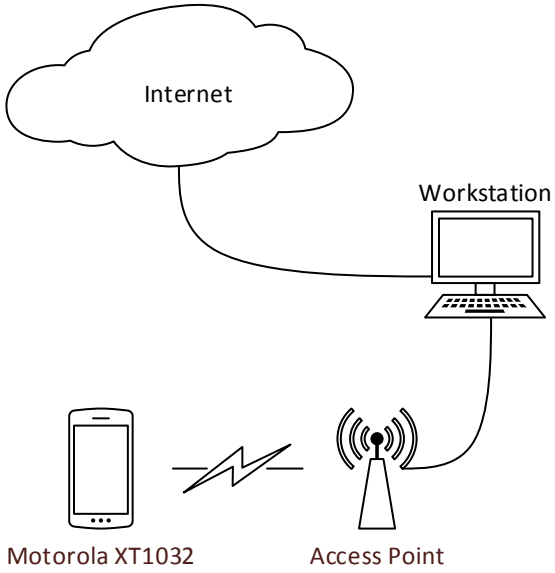


Fig. 1. Lab diagram of network used to perform the app audit and traffic analysis.

The network was set up as shown in Figure 1. The tests were conducted using a Motorola XT1032 smartphone running Android 4.4.2 in an isolated network segment that was designed to capture all traffic from the device. The sending of data over the cellular network was disabled using the smartphone's built-in settings, so that all data had to be sent over the Wi-Fi network it was connected to. The access point providing the Wi-Fi network was a Linksys WRT-320N access point connected to a network interface on the lab workstation. This setup enabled the analysis of the outbound data before it was routed to the Internet via another network interface on the same workstation. Wireshark [4] was used to perform the

Category	App Name	Number of Downloads
Communication	Whatsapp	500,000,000+
	Facebook Messenger	100,000,000+
	Skype	100,000,000+
	Viber	100,000,000+
	AntiVirus Security - FREE	100,000,000+
Games	Don't Tap the White Tile	10,000,000+
	100 Ballz	5,000,000+
	Candy Crush Saga	100,000,000+
	Slots Oz- slot machines	100,000+
	Clash of Clans	50,000,000+
Health & Fitness	Calorie Counter - MyFitnessPal	10,000,000+
	Strava	1,000,000+
	Runkeeper	10,000,000+
	A taste of Slimming World	500,000+
	Walk with map my walk	1,000,000+
Productivity	Super-bright LED Torch	100,000,000+
	Adobe Reader	100,000,000+
	Google Docs	1,000,000+
	Google Sheets	1,000,000+
	Dropbox	100,000,000+
Shopping	Ebay	50,000,000+
	Amazon	5,000,000+
	Groupon	10,000,000+
	Barcode Scanner	100,000,000+
	Wish-shopping made fun	10,000,000+
Random	Shazam	100,000,000+
	Battery Doctor	10,000,000+
	Tinder	10,000,000+
	JUST EAT	1,000,000+
	BBC Weather	1,000,000+
	Tippy Tap	100,000+
	High-Powered Torch	1,000,000+
	thetrainline	1,000,000+
	Speedtest.net	10,000,000+
	Retrica	10,000,000+

Table 1. List of 35 apps that were tested, grouped by category.

first round of tests that required passive analysis. For this research, the test device was not 'rooted' and no reverse-engineering was done on the apps that were being tested, to help ensure that no terms of service were violated.

For the second round of tests that required analysing network traffic encrypted using SSL, we used Mallory, an SSL interception proxy [2]. A trusted CA certificate was installed on the test device so that the Android OS would not reject the certificates being presented by Mallory. For the third round of tests, this trusted credential was removed from the test device to simulate a typical MITM attack where the attacker would not have had access to the victim device to install a fake certificate.

The test device came with factory installed software that also sent traffic over the network. Most of these services were manually disabled but others that were critical to the functioning of the system were left enabled. After disabling all non-critical services, a baseline test was done to determine the amount and type of "network noise" that was to be expected during our traffic analysis. The test device continued to sporadically communicate, mainly with a Google DNS server (*8.8.8.8*), NTP server (*pool.ntp.org*), another Google server (**.1e100.net*) and a Motorola server (**.svcmot.com*). This traffic was noted so it could be ignored if it was seen during traffic analysis.

During testing, each app was installed by itself on the test device to eliminate contamination and interference. After installation, each app was used for approximately 15 minutes to simulate normal use. During each test, all of the obvious features in the app were tested to trigger as much network traffic as possible and this allowed us to have a good idea of the traffic that the app would send throughout its lifetime.

For the traffic analysis, Wireshark and Mallory were configured to analyse traffic on standard HTTP (port 80) and HTTPS (port 443) ports. It is possible that traffic to non-standard ports was not analysed but apps and other pieces of software are usually designed to use standard ports to ensure the greatest flexibility in whatever network environment they are used.

5 Results

In Table 2, we summarize the results of the data gathered from the app audit. For brevity, we categorise various privacy leaks into broad categories. The VER (versions) category refers to data including one or more of device make/model, hardware version numbers or software version numbers. Version information is considered sensitive for two main reasons. Firstly, an attacker on the network may use version information to easily and quickly locate exploits for vulnerabilities that may be present in that specific hardware or software version. Secondly, given that users install a variety of apps on a variety of devices and may or may not keep their software versions up-to-date, an attacker could potentially be able to fingerprint (or narrow down the search space for) an individual device based on the type of device and versions of apps on that device.

The ID (device identifiers) category refers to data including one or more of telephony device ID (IMEI, MEID or ESN), telephone number or interface MAC address (Wi-Fi or Bluetooth for example). Some apps sent encoded data with variable names that suggested that the data contained some transformation of a telephony device ID. We did not attempt to identify whether the data actually contained a telephony device ID, thus only leakage of unencoded telephony device IDs are

App Name	HTTP Traffic Sent	SSL Traffic Sent	Leaks Over HTTP	Leaks Over SSL
Whatsapp	No	Yes	-	*
Facebook Messenger	No	Yes	-	GEN
Skype	Yes	Yes	-	-
Viber	Yes	Yes	GEN	VER, ID, GEN
AntiVirus Security - FREE	Yes	Yes	-	VER, ID
Don't Tap the White Tile	Yes	Yes	VER, ID, GEN	GEN
100 Ballz	Yes	Yes	VER	GEN
Candy Crush Saga	Yes	Yes	VER, GEN	-
Slots Oz- slot machines	Yes	Yes	VER, ID, GEN	VER, GEN
Clash of Clans	Yes	Yes	-	VER, GEN
Calorie Counter - MyFitnessPal	Yes	Yes	-	VER
Strava	Yes	Yes	VER	VER, ID, NARROW
Runkeeper	Yes	Yes	-	VER, GEN, NARROW
A taste of Slimming World	Yes	Yes	VER	-
Walk with map my walk	Yes	Yes	VER	VER, ID, GEN
Super-bright LED Torch	Yes	Yes	VER	VER, ID, GEN
Adobe Reader	No	Yes	-	VER
Google Docs	No	Yes	-	VER
Google Sheets	No	Yes	-	VER
Dropbox	No	Yes	-	*
Ebay	Yes	Yes	-	VER, GEN
Amazon	Yes	Yes	-	VER, GEN
Groupon	Yes	Yes	-	VER, GEN
Barcode Scanner	No	Yes	-	-
Wish-shopping made fun	Yes	Yes	-	GEN
Shazam	Yes	Yes	VER	VER, ID
Battery Doctor	Yes	Yes	ID	VER, ID, GEN
Tinder	Yes	Yes	-	VER, GEN, NARROW
JUST EAT	Yes	Yes	NARROW	-
BBC Weather	Yes	Yes	NARROW	-
Tippy Tap	Yes	Yes	VER	VER, ID, GEN
High-Powered Torch	Yes	Yes	VER	VER, ID, GEN
thetrainline	Yes	Yes	VER, ID, GEN	-
Speedtest.net	Yes	No	NARROW	-
Retrica	Yes	Yes	VER	GEN, NARROW

Table 2. Privacy leaks from the 35 apps that were tested. Hyphen (-) means no privacy leaks were observed. Asterisk (*) means no traffic was intercepted.

counted as a ID in Table 2. ID is considered private because this information, in the case of a telephony device ID or MAC address, is unique[1] to a particular device and may be used by an attacker to uniquely identify a device anywhere in the world. A telephone number is usually semi-permanent and is also considered private information because it can potentially identify an individual even across devices.

The GEN (general) category refers to data including one or more of country code, mobile carrier or city. This information is considered private because it can be used to get general information about a user such as their geographical area or the mobile carrier that they are subscribed to. The NARROW (narrowed) category refers to data including one or more of list of installed applications, post code or precise location. This information is considered private because it reveals narrowed information about a user that can identify their preferences (in the case of list of installed applications) or identify their precise geographic location.

From an analysis of Mallory's output, Whatsapp and Dropbox did not appear to be sending SSL traffic. Wireshark confirmed that these applications were terminating the SSL connections after they received the SSL certificates. Since Mallory was added as a trusted certificate signing authority on the device, an Android app would not be able to tell the difference between certificates if they relied on only the Android API. We suspect that these two applications are using custom-written procedures for handling SSL communication and may only be accepting particular certificates. Confirming whether this is the case is outside of the scope of this research. We ignore these applications when calculating percentages, where applicable, because we are unable to categorise the types of privacy leaks, if any, that these apps are responsible for.

From Table 2, it is apparent that 80% (28 of 35) of the apps that were audited sent some amount of information in plaintext (i.e. over HTTP), while 97% (34 of 35) of those same apps sent some amount of data encrypted using SSL. We also see that 64% (18 of 28) of the apps that sent unencrypted HTTP traffic were responsible for one or more types of privacy leaks. This means that an eavesdropper on the communication path will more often than not be able to capture some type of private information from a mobile device. Of the apps sending observable SSL traffic, 78% (25 of 32) of them leaked privacy in one way or another. It is moderately reassuring that more privacy leaks happen over an encrypted channel. This seems to indicate that app developers are, to some extent, wary of sending private information in plaintext and thus have taken steps to limit the visibility of private data. However, this is what was seen for the popular apps that were tested; many less popular apps cumulatively account for a large install base and they could be more privacy-eroding.

6 Discussion

It is interesting to note that although 97% (34 of 35) of the apps what were audited sent data encrypted using SSL, 80% (28 of 35) of those same apps sent data without any sort of encryption. An app may use both encrypted and unencrypted connections for the following reasons:

[1] Telephony device IDs and MAC addresses may be spoofed but we assume that spoofing has not been employed.

- App developers or the developers of libraries may choose not to implement SSL connections because of scalability issues. Establishing an SSL connection requires a longer handshake and this additional network overhead and latency may be noticeable to the user. Also, given that servers tend to process high volumes of requests, the additional time required to set up SSL connections may cause a significant increase in the number of concurrent connections that the servers must be designed to handle. This would require additional investment in infrastructure by providers without a direct return on their investment.
  Stevens et al. [14], discovered that all except one of the ad libraries that they analysed sent data in the clear; an observation that agrees with our explanation. A dating app that we audited, Tinder, sent GPS and other private information over an encrypted channel, but retrieved photos of dating matches in the clear using only HTTP. Since photos usually require more time and bandwidth to transfer, we believe this to be another case where the developers opted to sacrifice confidentiality to reduce server load and improve the user experience (in this case by reducing latency).
- SSL certificates are issued at a cost by issuing authorities and small app developers may be unwilling or unable to bear this cost and thus opt to send traffic without it being encrypted. During development, the developers may have opted to not use encryption to assist in debugging, and then forgot or did not bother to implement SSL since the app was already working as intended.

The above explanations are only suggestions of potential reasons for the observed behaviour; we leave confirmation of the actual reasons to future work.

6.1 Apps that fail to handle SSL certificates properly

After successfully intercepting SSL traffic using Mallory, the user-installed credential that allowed the Android device to accept the Mallory-signed certificates was removed. The apps were then retested to determine whether they would accept SSL certificates that were not properly signed by a trusted authority. Worryingly, a number of the apps that were audited still continued to send SSL traffic even when presented with certificates that did not have a proper chain of trust.

A custom Android app was written to confirm that the Android OS was properly identifying when an unsafe certificate was presented. Without the trusted credential installed on the device, the custom app was correctly throwing a *SSLHandshakeException* when Mallory presented its fake certificates. This suggested that the developers of the apps that continued to send traffic, wrote their code in such a way that circumvented the integrity protection built into the SSL certificate system.

It is possible that the developers of these apps wrote them in this way for development purposes. During development and testing, self-signed and other untrusted certificates are sometimes used to reduce costs and the time it takes to test the apps and servers. The developers could have forgotten to rewrite the code, after testing, to ensure that the app was once again able to properly identify certificates that did not have a complete chain of trust. The apps we identified behaving in this way were Battery Doctor, Candy Crush Saga, Clash of Clans, Don't Tap the White Tile, Retrica and Runkeeper. These apps revealed private information from all 4 of our categories: VER, ID, GEN and NARROW; from software versions to IMEI number to precise geographic location.

A clever attacker, having a list of apps that did not handle untrusted certificates properly, could target MITM attacks to traffic from only these susceptible apps. In

this way, they would be able to pilfer private data from a device unbeknownst to the device's user.

6.2 Apps that only use specific certificates

Dropbox and Whatsapp sent no SSL traffic according to Mallory. Wireshark was used to analyse the network traffic to see if the reason could be identified. Both apps seemed to terminate the connection after receiving the Mallory-generated SSL certificate. Since a trusted credential was installed on the device at this time, the Android OS should have accepted the certificate. This leads us to believe that the developers of these applications were using separate certification validation routines within the app code.

This is not unusual behaviour, since software such as web browsers usually store their own list of trusted certificate signing authorities instead of relying on the ones used by the system. While this certificate validation strategy would prevent some kinds of MITM attacks, there is a sacrifice in versatility of deployment for the app. For example, these apps would not work by default on networks where SSL interception is employed to allow a firewall to inspect network traffic. If our suspicion is correct, these apps are at a further disadvantage because, unlike the web browsers that have separate trusted authorities, these apps have no way for the end-user to easily add additional certificate signing authorities. Given the maturity of SSL, we would not recommend app developers to use separate certificate validation strategies unless their apps have a very specific need to do so.

6.3 Apps that only send encrypted data

The following apps from our audit list did not send any observable traffic unencrypted: Whatsapp, Facebook Messenger, Adobe Reader, Google Docs, Google Sheets, Dropbox and Barcode Scanner. This is the recommended behaviour for apps and it is commendable that the more well-established companies that market (all except one of) these apps lead by example. However, the other 80% (28 of 35) of apps that send some traffic in the clear are very widely installed by the consumer base and care must be taken to reduce potential privacy leaks that come from these applications.

6.4 What attackers can learn from unencrypted data

An attacker can learn a number of things from unencrypted traffic. For example, they can identify the make and model of the mobile device as well as the version of the OS or apps that are installed. The version information can help the attacker to identify exploits that may be used against the particular device or application. Given that the Android OS is customised by vendors for their particular devices, critical security patches may take longer to get to the end-user and thus an attacker could have greater opportunity to identify and exploit vulnerable devices on the network.

An attacker can also learn what apps their target uses, and how often and when they use them. This data can be used to profile a target and could potentially be applied to help identify a target device based on its traffic and app usage pattern. To help make the point that an attacker knowing what apps a person uses may be undesired, consider the following example. An investigative journalist or private

investigator connects to the same Wi-Fi connection (provided by a coffee shop for example) as their target. If the target was a married, middle-aged politician, it might be valuable for the attacker to discover that the target has several dating apps installed on their mobile device. If Tinder is one of these dating apps, then, as we discovered during our audit, the attacker would be able to passively obtain pictures of the potential matches that the target was receiving. These images could then suggest the types, ages and gender of persons that our target was interested in.

An attacker can also perform an MITM attack more easily on unencrypted traffic. Thus they could alter incoming or outgoing data from a mobile device to achieve whatever objective they were aiming for.

6.5 What attackers can learn from encrypted data

Even if all apps encrypted their data fully before sending, that would only serve to limit the approaches that an attacker could take to extract information. As Stöber et al. discovered, even encrypted traffic still leaks side channel information such as timing and data volume. They found that fingerprinting traffic can allow the unique identification of a smartphone with up to a 90% chance of success [15].

In addition to fingerprinting encrypted traffic, and attacker could also infer the apps that were installed based on the endpoints of encrypted connections. Different apps connect to various servers and these remote addresses are visible in the connections. An attacker may fingerprint individual applications and use this knowledge to uniquely identify various apps that have been installed on a mobile device. From our example with the middle-aged politician above, even if the traffic was fully encrypted, an attacker may still be able to identify that the target was using dating apps as well as how frequently they were using them.

As we discussed above, badly written apps could encrypt traffic but still be vulnerable to MITM attacks because of the way they were written to handle SSL certificate trust verification. This would allow an attacker to analyse the supposedly secure data and gain valuable insights about their target because of bad or careless programming.

6.6 Advanced MITM attacks on mobile devices

A skilled attacker could successfully craft a more sophisticated attack using artefacts unique to mobile devices. As we discovered on our test device and as shown in Figure 2, if the user uses the default browser (Google Chrome in the case of our test device) to navigate to a link that contains a valid certificate file, a dialogue box will pop up immediately and prompt the user to install the certificate as a trusted certificate.

The dialogue does not give any indication of the potential consequences of accepting the certificate. If an attacker can successfully trick a user into installing a rogue certificate, then most, if not all, encrypted transmissions coming from that device may then be intercepted. Performing such an attack on a mobile device may be easier than on a traditional computer for two reasons:

1. As the literature [13][7] points out, users are less security-conscious on mobile devices. This could be because they trust the app ecosystem or because mobile devices are less targeted than traditional computers.

2. The mobile device usually has a smaller screen and this limits the amount of space that developers may be willing to dedicate to warning messages. Also, many mobile browsers hide the URL bar (to save space) when it is not directly being used. An attacker could thus use a MITM attack on an unencrypted connection and redirect the user to a malicious web page or rogue certificate without the user being immediately aware of what is happening.

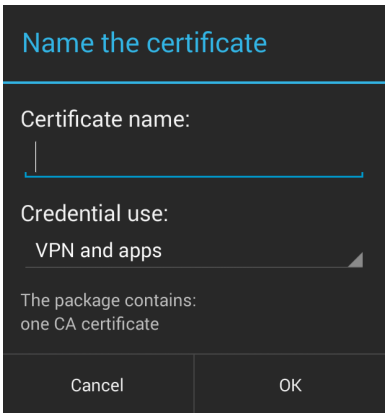


Fig. 2. Screenshot of the dialogue presented to a user after navigating to a potentially dangerous certificate via the default web browser. The dialogue does not indicate the potential consequences of installing the certificate.

With proper crafting of websites and adequate social engineering, an attacker could trick the user into installing a rogue certificate on their device. If this attack is successful, then not only the traffic from all the apps on the device would be available, but also the user's login credentials used to access popular services such as social networking, email and banking. An attacker with a device spoofing a popular Wi-Fi network (such as that from a coffee shop or a university campus) could potentially target hundreds or thousands of users simultaneously using this hybrid attack.

6.7 Potential Solutions to the Problems Identified

Below we suggest potential solutions to the problems outlined above:

Encrypt all traffic from apps Encrypting all traffic from apps increases the amount of investment required by an attacker. They would no longer be able to passively overhear traffic coming from apps. They would also not be able to easily perform a MITM attack on the traffic from the device. It is also necessary to ensure that apps using SSL properly check certificates and reject those that do not have an appropriate chain of trust. Fully encrypting data helps alleviate the problem but it may not be widely implemented by developers for the reasons discussed in Section 6.

Use onion routing to obfuscate endpoints Onion routing, where messages are encrypted and sent through various network nodes, can be used to help defeat an attacker using side channel data (such as traffic endpoints) to attempt to enumerate the apps that are installed on a device. Onion routing increases computation and transmission overhead and thus may be infeasible for use on mobile devices where battery longevity is critical. A potentially better solution that would better preserve battery life is to have the mobile device send all its traffic through a Virtual Private Network (VPN) to a server that then does the onion routing.

User education User education is also of critical importance, especially on mobile devices where the screen size is smaller and warning messages may be very concise or non-existent. Users need to be made aware that the same attacks that are carried out on traditional computers can also target mobile devices. The user should be reminded that their mobile device contains much personal information and that they use it for critical and sensitive tasks (such as email, shopping and banking) just like their traditional computer.

7 Conclusion and Future Work

This research has highlighted a number of areas where privacy is leaked from mobile devices. Given that these mobile devices may store a trove of sensitive data it is pertinent to examine ways in which privacy leaks can be reduced or eliminated. We identify attacks that can be employed against mobile devices and we suggest mitigation strategies to alleviate the problems identified. Strategies such as encryption (and the proper handling of certificates) by apps, onion routing and user education can all be used to reduce the attack surface and increase the investment required by an attacker to pull of a successful attack.

No one solution is a panacea, and as mobile devices become more pervasive and more accessible to low-budget consumers, the landscape will undoubtedly continue to evolve and new and dynamic protection systems will need to be developed to keep the end user safe. We plan to extend our work by doing large-scale, and potentially automated, audits of apps across multiple app stores and operating systems to gain a better understanding of the collective challenges that consumers and security practitioners currently face. We will continue to identify clever attacks that can be aimed at end-users and develop novel technologies to combat these attacks. What is for sure is that security and privacy is an ongoing process in the information age and many open problems will continue to emerge to keep security researchers occupied into the foreseeable future.

References

1. Downloads on Google Play - AppBrain, http://www.appbrain.com/stats/android-app-downloads
2. Mallory: Transparent TCP and UDP Proxy - Intrepidus Group - Insight, https://intrepidusgroup.com/insight/mallory/
3. Monetise, Promote and Analyse your apps with AdMob, http://www.google.co.uk/ads/admob/
4. Wireshark, http://www.wireshark.org/

5. Beresford, A.R., Rice, A., Skehin, N., Sohan, R.: MockDroid: trading privacy for application functionality on smartphones. In: Proceedings of the 12th workshop on mobile computing systems and applications. pp. 49–54. ACM (2011)
6. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: OSDI. vol. 10, pp. 1–6 (2010)
7. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: User attention, comprehension, and behavior. In: Proceedings of the Eighth Symposium on Usable Privacy and Security. p. 3. ACM (2012)
8. Gartner: Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments Are On Pace to Grow 6.9 Percent in 2014 (March 2014), http://www.gartner.com/newsroom/id/2692318
9. Gibler, C., Crussell, J., Erickson, J., Chen, H.: AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale. In: Trust and Trustworthy Computing, pp. 291–307. Springer (2012)
10. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In: Proceedings of the 18th ACM conference on Computer and communications security. pp. 639–652. ACM (2011)
11. Kim, J., Yoon, Y., Yi, K., Shin, J., Center, S.: ScanDal: Static analyzer for detecting privacy leaks in android applications. MoST (2012)
12. Mann, C., Starostin, A.: A framework for static detection of privacy leaks in android applications. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing. pp. 1457–1462. ACM (2012)
13. Mylonas, A., Kastania, A., Gritzalis, D.: Delegate the smartphone user? Security awareness in smartphone platforms. Computers & Security 34(0), 47 – 66 (2013), http://www.sciencedirect.com/science/article/pii/S0167404812001733
14. Stevens, R., Gibler, C., Crussell, J., Erickson, J., Chen, H.: Investigating user privacy in Android ad libraries. In: Workshop on Mobile Security Technologies (MoST). Citeseer (2012)
15. Stöber, T., Frank, M., Schmitt, J., Martinovic, I.: Who Do You Sync You Are?: Smartphone Fingerprinting via Application Behaviour. In: Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks. pp. 7–12. WiSec '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2462096.2462099
16. Yang, Z., Yang, M., Zhang, Y., Gu, G., Ning, P., Wang, X.S.: Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 1043–1054. ACM (2013)