iPhone Privacy

Nicolas Seriot[*]
http://seriot.ch

Black Hat DC 2010
Arlington, Virginia, USA

Abstract

It is a little known fact that, despite Apple's claims, any applications downloaded from the App Store to a standard iPhone can access a significant quantity of personal data.

This paper explains what data are at risk and how to get them programmatically without the user's knowledge. These data include the phone number, email accounts settings (except passwords), keyboard cache entries, Safari searches and the most recent GPS location.

This paper shows how malicious applications could pass the mandatory App Store review unnoticed and harvest data through officially sanctioned Apple APIs. Some attack scenarios and recommendations are also presented.

Keywords: Apple, iPhone, Security, Privacy, App Store, Malware.

[*]Nicolas Seriot is a software engineer in Switzerland. He has taught iPhone development at Sen:te and is now a scientific collaborator at School of Business and Engineering Vaud (HEIG–VD). Nicolas holds a Master's degree in Economic crime investigation.

Contents

1 Introduction

1.1 Background facts

This subsection presents some quick facts and figures for people unfamiliar with the iPhone.

1.1.1 Market share

Apple introduced the iPhone on June 29, 2007. iPhone worldwide market share of smartphone sales to end users was 17% in Q3 2009[1]. In late 2009, 34 millions iPhones had beed sold[2].

1.1.2 App Store

Developers distribute iPhone applications through Apple's App Store, which had more than 100,000 applications available[3] in late 2009. As of January 5, 2010 three billion applications have been downloaded[4] in 70 countries worldwide.

1.1.3 Jailbreak

In its standard configuration, the iPhone only runs software cryptographically signed by Apple. However, iPhone hackers have found ways to circumvent this limitation by creating a modified version of the iPhone OS that will run any code. Installing such a firmware is called "jailbreaking". Most "jailbroken" devices feature an SSH server and enable the root login at the same time. According to Intego[5] about 6–8% of all iPhones are "jailbroken".

1.2 Motivation

1.2.1 Press reports

The second half of 2009 has been a huge commercial success for Apple, but at the same time numerous reports have surfaced about privacy issues with the iPhone.

> *Customers of ID Mobile's MogoRoad iPhone application are complaining that they're getting sales calls from the company, a process which turns out to be technically a piece of cake.*
>
> The Register, 30th September 2009
>
> `http://www.theregister.co.uk/2009/09/30/iphone_security/`

[1]`http://www.gartner.com/it/page.jsp?id=1224645`
[2]`http://www.apple.com/pr/library/`
[3]`http://www.apple.com/pr/library/2009/11/04appstore.html`
[4]`http://www.apple.com/pr/library/2010/01/05appstore.html`
[5]`http://blog.intego.com/2009/11/11/`

A maker of some of the most popular games for the iPhone has been surreptitiously collecting users' cell numbers without their permission, according to a federal lawsuit filed Wednesday.

The Register, 6th November 2009

http://www.theregister.co.uk/2009/11/06/iphone_games_storm8_lawsuit//

iPhone owners in Australia awoke this weekend to find their devices targeted by self-replicating attacks that display an image of 1980s heart throb Rick Astley that's not easily removed.

The Register, 8th November 2009

http://www.theregister.co.uk/2009/11/08/iphone_worm_rickrolls_users/

In a world where lost or stolen laptops and phones scream at us from newspaper headlines and data breaches invite multi-million dollar government fines and lawsuits, do we want a smartphone whose insecurity is well documented? In our judgment, no.

Sharon D. Nelson and John W. Simek, Sensei Enterprises, 2009 [4]

Mark Bregman, Symantec's executive vice president and CTO, highlighted mobile security as a critical area to monitor in the new year, in which "more attackers will devote time to create malware to exploit these devices". In a separate report, Symantec noted that Apple's products, including its popular iPhone, will face increased attacks from cybercriminals.

ZDNet Asia, 9th December 2009

http://www.zdnetasia.com/news/software/0,39044164,62059902,00.htm

Should people care? Should they be scared? Can users trust iPhone applications because they are reviewed by Apple? What about corporate environments? The aim of the work presented in this paper is to get facts and a clear view of iPhone privacy issues, in order to help consumers and professionals taking sensible and well-informed decisions.

1.2.2 Outline of the paper

This paper starts by defining privacy and presenting an overview of past iPhone privacy issues.

Next, it specifically considers the unmodified devices and examines what sensitive data may be compromised by an application downloaded from the App Store. Because seeing is believing, the source code of a proof-of-concept malicious application was made available publicly.

This paper goes on to explain how a malicious application could be crafted to fool Apple's mandatory reviews in order to be accepted on the App Store. Finally, it discusses several attack scenarios, tries to suggest improvements and delivers basic recommendations.

This paper doesn't try to compare other mobile platforms such as Android, Windows Mobile or BlackBerry. It also doesn't discuss data harvesting on desktop computers.

2 Privacy, spyware and law

2.1 Privacy and spyware

According to the Merriam-Webster's dictionary, privacy is *the quality or state of being apart from (...) observation*. In the context of spyware applications, we can consider privacy more narrowly as *personal data confidentiality*.

By spyware, I mean an application that gathers personal data without the user's knowledge and consent. So what is personal data? At the end of the day, the definition that matters is the legal one. Of course, it depends on the country one lives in. Because the author of this paper lives in Switzerland, let us have a quick look at Swiss law, which is quite strict regarding protection of privacy.

Note that this law is only applicable if the spyware author is either resident in Switzerland, or if the government can successfully negotiate extradition to Switzerland. In the European Union, the extradition is allowed by the Convention on Cybercrime[6].

2.2 Swiss law

2.2.1 Personal data

The Swiss Federal Constitution states that *everyone has the right to be protected against the misuse of their personal data* (Art. 13 al. 2). Personal

[6] http://conventions.coe.int/Treaty/EN/Treaties/html/185.htm

data are then defined in a specific law called the Federal Act on Data Protection (FIDP), which defines personal data as all information relating to an identified or identifiable person (FIDP Art. 3 lit. a).

2.2.2 Personality profile

The same law defines a collection of personal data as a "personality profile". It permits an assessment of the essential characteristics of the personality of a natural person (FIDP Art. 3 lit. d). Personality profiles are especially protected and strictly regulated (eg. FIDP Art. 11 lit. a al. 3).

2.2.3 Spyware authors

Now, where are the legal responsibilities? Spyware authors may be jailed for up to three years and have to pay big fines for "obtaining personal data without authorization" (Criminal Code Art. 179 novies). Also, applicable civil laws can be used to impose hefty fines (Breaches of privacy FIDP Art. 12.2, Breach of obligations to provide information or to cooperate FIDP Art. 34). This is scarcely applied, though.

2.2.4 End users

End users are protected from over reaching End User License Agreements (EULAs). Arbitrary provisions that have been "snuck into" EULAs are void (Civil Code Art. 2 al. 1). Thus, the EULA cannot simply state that you agree to send your personal data to bad guys if you do not. There must be a real mutual agreement, ruling out the use of potentially misleading terms (Civil Code).

2.2.5 Technical staff

In case of damages, civil liability may apply to technical staff if the plaintiff can prove that an organization failed to protect confidential data properly (FIDP-O Art. 8, 9, 10, 11, 12). Under this law, liability could extend all the way to Apple itself. For more detail, see this document [2] by the Swiss Federal Data Protection and Information Commissioner.

3 History of iPhone privacy concerns

This section presents the main privacy issues that have been encountered on the iPhone. Figure 1 summarizes them on a timeline.

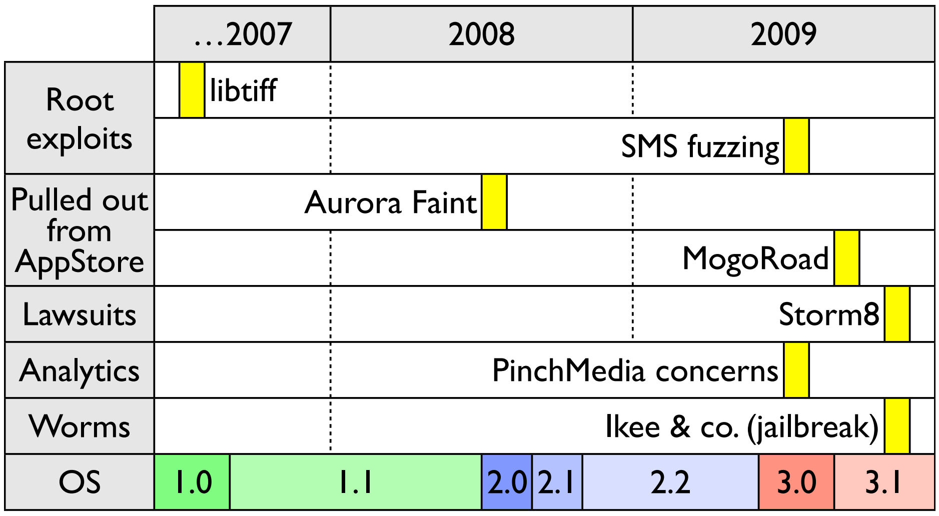


Figure 1: Timeline of the main past privacy issues

3.1 Root exploits

Here are two well known root exploits on the iPhone. The vulnerabilities were quickly patched by Apple, but could have been exploited to steal private data.

3.1.1 libtiff

July 2007. The first exploit was due to multiple buffer overflows[7] discovered in libtiff by Tavis Ormandy. The vulnerable libtiff version was used by the Apple's ImageIO framework. The simple opening of a maliciously crafted TIFF image could lead to arbitrary code execution, as demonstrated[8] by Rik Farrow. Apple patched[9] this vulnerability in iPhone OS 1.1.2.

3.1.2 SMS fuzzing

July 2009. This exploit was presented at Black Hat USA 2009 by Charlie Miller and Collin Mulliner. The researchers presented an iPhone vulnerability[10] that could allow a hacker to seize control of the phone through maliciously crafted SMS messages. The vulnerability was patched[11] in iPhone OS 3.0.1.

[7] CVE-2006-3459, CVE-2006-3461, CVE-2006-3462, CVE-2006-3465

[8] http://www.fastcompany.com/multimedia/2007/11/hacking-the-iphone.html

[9] http://support.apple.com/kb/HT2170

[10] http://securityevaluators.com/content/news/index.jsp?topic=iphone_sms_2009

[11] http://support.apple.com/kb/HT3754

3.2 Personal data harvesting

3.2.1 Aurora Feint

In July 2008, the popular iPhone game *Aurora Feint* was the first application to be pulled from the App Store due to privacy concerns. The game would upload all the contacts stored in the iPhone to the developer's server, allegedly to discover if any of the user's friends also play that game.

3.2.2 MogoRoad

In September 2009, the Swiss road traffic information application MogoRoad was pulled from App Store after users complained they got sales calls from the company. MogoRoad is back on App Store after Mogo's explanations[12].

3.2.3 Storm8 complaint

In November 2009, a federal lawsuit was filed in California against iPhone applications editor Storm8, whose games had already been downloaded more than 20 million times. The games were harvesting the user's phone number[13] without encryption. Since then, Storm8 games have stopped collecting the users' phone numbers.

3.2.4 Pinch Media

Pinch Media[14] is a free analytics framework used by many iPhone developers. It collects anonymous usage data from mobile phone applications and could be compared to Google Analytics.

In July 2009, some bloggers started to raise serious concerns[15] about Pinch Media, claiming that iPhone users were being tracked by some applications without their knowledge and without the possibility of opting out. According[16] to Pinch Media, the collected data are:

– a unique hardware identifier
– the model of your phone and operating system
– the application's name and version
– the result of a check to see if the device has been jailbroken
– the result of a check to see if the application has been stolen
– the length of time the application was run

[12]http://www.mogo.ch/presse/ID_MOBILE_COMMUNICATE_MOGOROAD_EN.pdf

[13]http://www.sfgate.com/cgi-bin/blogs/ybenjamin/detail?entry_id=46236

[14]http://www.pinchmedia.com/

[15]http://i-phone-home.blogspot.com/2009/07/pinchmedia-anatomy-of-spyware-vendor.html

[16]http://www.pinchmedia.com/blog/pinch-media-user-privacy-and-spyware/

  - if the user explicitly agrees to share it, the user's location
  - if the application uses Facebook Connect, the gender & age of the user

3.3 Worms on jailbroken devices

November 2009 saw an important wave of worm attacks targeting jailbroken iPhones. All of them exploit the fact that very few users bother to change the default root password (`alpine`) after jailbreaking their iPhone and installing an SSH server.

3.3.1 Ikee

Ikee is the first known iPhone worm. It changes the iPhone's wallpaper and displays a photograph of 1980s singer Rick Astley with the words *Ikee is never gonna give you up.* It was written by a 21-year old australian programmer, who was subsequently hired by the Australian iPhone development company *mogeneration.*

3.3.2 Dutch 5 € ransom

This worm locked the screen with the following message: *Your iPhone's been hacked because it's really insecure! Please visit doiop.com/iHacked and secure your iPhone right now!*, until the user had paid a 5 € ransom on a PayPal account. The Dutch hacker has now taken down his PayPal, returned the money he earned and published free instructions on how to remove the backdoor.

3.3.3 iPhone / Privacy.A

This worm steals personal data but, unlike the previous worms, does not reveal its presence.

3.3.4 Ikee.B / Duh

This worm is highly pernicious. It connects to a Lithuanian master, like a traditional botnet node. It changes the root password into "`ohshit`", steals private data and attempts to infect other hosts. The worm also tries to exploit ING Direct bank's two-factor authentication by using SMS. This worm was analyzed by SRI International [5].

3.4 iPhone forensics

Physical access to any device means that pretty much everything can be compromised, with the notable exception of passwords, which are stored encrypted in the phones Keychain.

Jonathan Zdziarski [3] showed that both the passcode and the disk encryption could be bypassed and the full iPhone disk image could be retrieved unencrypted by an investigator with the proper tools. This led an unnamed mid-sized US law firm (50 plus lawyers) to completely ban the iPhone[17].

An interesting white paper [6] on iPhone forensics was also released in March 2009 by viaForensics.

3.5 Security solutions editors

As a result of the late 2009 worms, Apple got a lot of bad press, much of which was unjustified. Some people criticize Apple for having an insecure configuration, whereas in fact worms only target jailbroken devices that have an SSH server *and* root access with the default password. It has also been suggested that jailbroken iPhone are more secure than stock devices because users can theoretically install firewalls.

Neither firewalls nor antivirus products are available for stock iPhones, from either Apple or third parties, so, not surprisingly, security software editors are among the most vocal critics.

Chester Wisniewski from Sophos writes on his blog that *iPhones are not ready for the business environment*[18] because they can be jailbroken. This is somewhat unfair. Would anybody claim that Linux is not ready for business because you can exploit a weak default root password on an SSH server?

Thus, F-Secure[19] and Intego[20] argue that Apple should modify the iPhone OS design and allow background running applications with a broader access to the OS, which would have the significant effect of allowing vendors to offer security solutions to iPhone users.

4 Writing spyware for the iPhone

4.1 Methodology

Now imagine that we would like to write a spyware for the iPhone. It would look like a breakout game and actually play breakout but, at the same time, silently harvest personal data and send them to a remote server.

[17]`http://ridethelightning.senseient.com/2009/12/my-entry.html`
[18]`http://www.sophos.com/blogs/chetw/g/2009/11/21/malicious-iphone-worm-loose/`
[19]`http://www.itpro.co.uk/618154/security-firms-cannot-protect-the-iphone-from-threats`
[20]`http://www.theregister.co.uk/2009/11/25/iphone_anti_malware/`

In order to reach the App Store, the spyware can't normally use private APIs, because this is forbidden by Apple and checked during the mandatory review process. It must also run on a non-jailbroken iPhone, namely on the original iPhone OS version 3.1.2, which is the latest version available at the time of writing.

Let's look for entry points to steal personal data within these constraints. We consider neither hardware attacks — see [3] on this matter — nor root exploits. Also, we don't target the information voluntarily submitted by users, such as Facebook or Twitter profiles, although a real-life spyware would certainly be interested in these data. It would be really easy to harvest Twitter or Facebook details, by offering to upload high scores to social networks.

4.2 Entry points

4.2.1 Cell Number

The first and easiest item of personal data to collect is the user's phone number. This number is entered in iTunes when the phone is first connected. The good news is that you can change it anytime in iPhone Settings. The number is stored in the file `.GlobalPrefrences.plist`. Listing 1 shows hot to retrieve the number programmatically.

Listing 1: Retrieve the cell phone number

```
NSDictionary *d = [NSUserDefaults standardUserDefaults];
NSString *phone = [d valueForKey:@"SBFormattedPhoneNumber"];
```

Novice iPhone programmers sometimes think that they need the cell number, when in fact they just need some kind of device-specific identifier. In this case, they should use the Objective-C method `-[UIDevice uniqueIdentifier]`. Note that this unique identifier can also raise privacy concerns, as discussed in section 7.1.6.

4.2.2 Address Book

Another way to collect personal data is through the Address Book API. It turns out that the full Address Book is readable without the user's knowledge or consent. It contains names, users' phone numbers and email addresses, but also a "notes field", in which many Mac users store sensitive data such as door codes or bank accounts. These notes are synchronized with the user's computer and may be harvested on the iPhone. Moreover, a spyware can edit the Address Book, that is add, change or delete any record, without the user's knowledge or consent.

This surprising state of affairs appears to reflect a deliberate decision by Apple. This API is very similar to the one on the Mac, but it lacks the "Me" record, as if Apple had sought to prevent the iPhone owner identification.

Spammers can easily retrieve all the email addresses, and more sinister scenarios are all too readily imaginable. For example, if a rogue application alters the email addresses, there is a great potential for harm, because the Mail application displays the contact's full name only, and a confidential email could be sent to the wrong address (see attack scenarios section 6).

The Address Book issue is one of the most serious for corporate environments.

4.2.3 File system

Next, we consider the data that can be read on the iPhone filesystem. A sandboxing mechanism limits access to other application's data. Third-party applications are installed in `/private/var/mobile/Applications/` and are prevented from seeing each other or accessing specific locations, such as the Music Library for instance.

The sandboxing mechanism is implemented at the kernel level and described by a set of rules shown in the file `SandboxTemplate.sb` (appendix A). See `man sandbox` on Mac OS X for more details.

About the sandboxing mechanism, Apple writes:

> *Applications on the device are "sandboxed" so they cannot access data stored by other applications.*
>
> *In addition, system files, resources, and the kernel are shielded from the user's application space.*
>
> Apple, iPhone in Business: Security Overview [1]

It turns out that, despite sandboxing, numerous system and application preference files are in fact readable (see listing 2) by downloaded applications, and some of them contain personal data. This concerns primarily preference files in plist format, but is not limited to them. The file system can be browsed and the content of these files viewed using the open-source file system browser FSWalker[21].

[21] http://fswalker.googlecode.com/

Indeed, as suggested by the comments found in the file in the appendix A, the sandbox rules are far too loose. This appears to be a case of security being sacrificed in the rush to get a product to market.

Note that our spyware doesn't even need undocumented APIs to read these files. Also, the user is never asked nor informed of the application poking the file system. He cannot prevent it.

Listing 2: Printing the paths readable on the file system

```
- (void)printReadablePaths {
    NSFileManager *fm = [NSFileManager defaultManager];
    NSDirectoryEnumerator *dirEnumerator = [fm enumeratorAtPath:@"/"];
    NSString *path = nil;

    while(path = [dirEnumerator nextObject]) {
        if([fm isReadableFileAtPath:path]) {
            NSLog(@"-- %@", path);
        }
    }
}
```

4.3 Introducing SpyPhone

At this point, we know where to find personal data, including the phone number, the Address Book contents and several other pieces of data readable on the file system. A spyware would want to gather the potentially valuable information from all these sources. SpyPhone[22] is an open-source[23] proof of concept "that does just that". Figure 2 shows the files actually read by SpyPhone. Let's have an overview of the data the rogue application can collect. You may want to look at the source code for details.

The spyware can access the 20 most recent Safari searches (figure 3) and YouTube history (figure 4). It can access the email account parameters (figure 6), such as the full name of the user, the email address, as well as the host and login. The password is kept safe in the Keychain. The email content itself is not accessible, thanks to correct sandbox rules.

The spyware can access data about the phone (figure 5) itself, starting with the phone number, as discussed in section 4.2.1. The spyware can also read the iPhone UUID (through a documented API), the ICCID (SIM card serial number) and the IMSI (International Mobile Subscriber Identity), making it possible to track users even when they change their device. IMSI reveals the country and the mobile operator.

[22]http://github.com/nst/spyphone/

[23]You still need an iPhone developer certificate to install it on a non jailbroken iPhone.

```
/var/mobile/Library/Keyboard/
/var/mobile/Library/Preferences/com.apple.accountsettings.plist
/var/mobile/Library/Preferences/com.apple.commcenter.plist
/var/mobile/Library/Preferences/com.apple.mobilephone.settings.plist
/var/mobile/Library/Preferences/com.apple.mobilephone.plist
/var/mobile/Library/Preferences/com.apple.mobilesafari.plist
/var/mobile/Library/Preferences/com.apple.preferences.datetime.plist
/var/mobile/Library/Preferences/com.apple.weather.plist
/var/mobile/Library/Preferences/com.apple.youtube.plist
/var/mobile/Library/Preferences/com.apple.Maps.plist
/var/mobile/Media/DCIM/
```

Figure 2: Paths actually read by SpyPhone

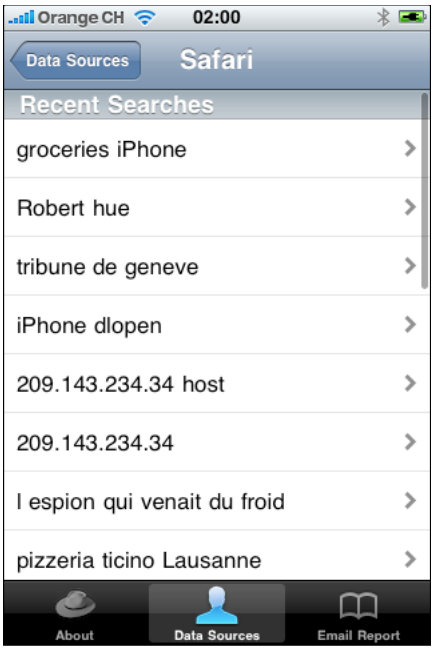
Figure 7 shows the email addresses stored in the Address Book database. It is also possible to display any other field, as discussed in 4.2.2.

The keyboard cache contains all the words ever typed on the keyboard, except the ones entered in password fields. This is supposed to help autocompletion but this mechanism effectively acts as a key-logger, storing potentially private and confidential names and numbers. Figure 9 shows the structure of a keyboard cache file containing the dictionary. There is one cache file per language. The keyboard cache can be reset in iPhone Settings.

Any application has read and write access to the DCIM directory that contains the photos stored in the iPhone. By default, these photos are tagged with the GPS coordinate. By listing (figure 10) the date and location of each one, SpyPhone can thus keep detailed track of a user's physical movements. These data can then be mapped (figure 11). Note that SpyPhone can also send the photo itself by reading it on the file system, where a program would normally be able to access only one image at a time through the UIImagePickerController class, chosen by the user, and without EXIF and GPS data.

The first time an application needs to access the GPS location, it prompts the user. But a spyware could also gain a pretty accurate estimate of the user's location without enabling the GPS. It could simply read the previous position in the Maps application preferences (figure 12). It could also know which areas you are in simply by reading the weather cities. A marketer rarely needs a more precise location.

Another file keeps track of every time you join a Wifi network. Based on these logs, it is possible to gain insights into the user recents' whereabouts.

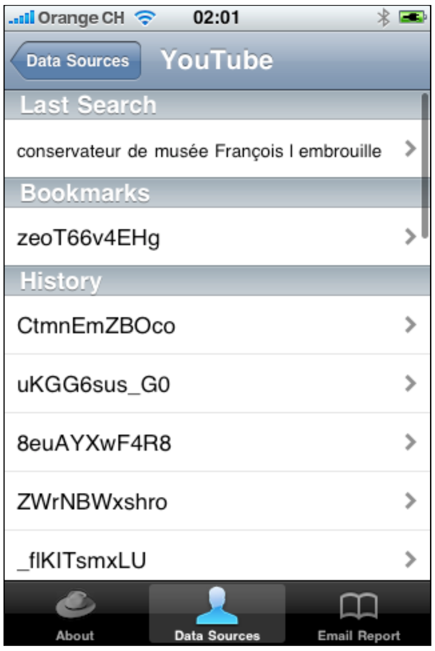

Orange CH 02:00

Data Sources **Safari**

Recent Searches
- groceries iPhone
- Robert hue
- tribune de geneve
- iPhone dlopen
- 209.143.234.34 host
- 209.143.234.34
- l espion qui venait du froid
- pizzeria ticino Lausanne

About | Data Sources | Email Report

Figure 3: Safari Searches History


Orange CH 02:01

Data Sources **YouTube**

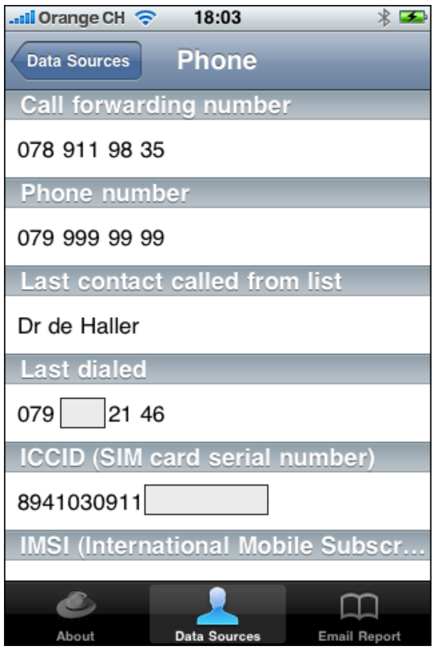
Last Search
- conservateur de musée François I embrouille

Bookmarks
- zeoT66v4EHg

History
- CtmnEmZBOco
- uKGG6sus_G0
- 8euAYXwF4R8
- ZWrNBWxshro
- _fIKITsmxLU

About | Data Sources | Email Report

Figure 4: YouTube History


Orange CH 18:03

Data Sources **Phone**

Call forwarding number
078 911 98 35

Phone number
079 999 99 99

Last contact called from list
Dr de Haller

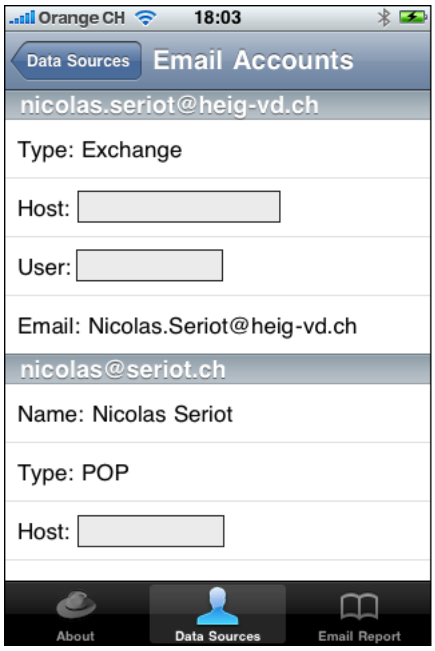
Last dialed
079 21 46

ICCID (SIM card serial number)
8941030911

IMSI (International Mobile Subscr…

About | Data Sources | Email Report

Figure 5: Phone and Hardware


Orange CH 18:03

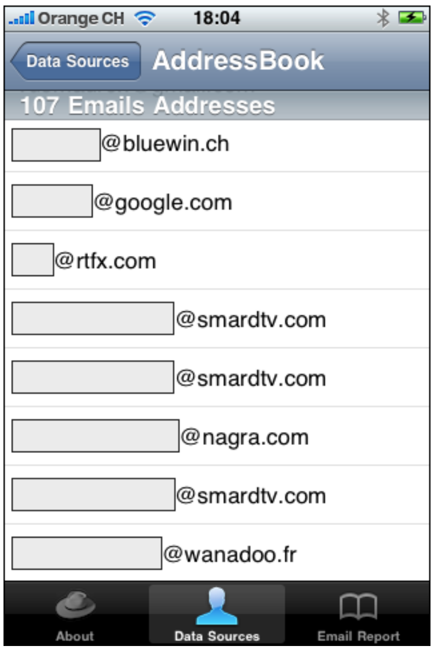
Data Sources **Email Accounts**

nicolas.seriot@heig-vd.ch
- Type: Exchange
- Host:
- User:
- Email: Nicolas.Seriot@heig-vd.ch

nicolas@seriot.ch
- Name: Nicolas Seriot
- Type: POP
- Host:

About | Data Sources | Email Report

Figure 6: The email accounts



AddressBook

107 Emails Addresses

[redacted]@bluewin.ch

[redacted]@google.com

[redacted]@rtfx.com

[redacted]@smardtv.com

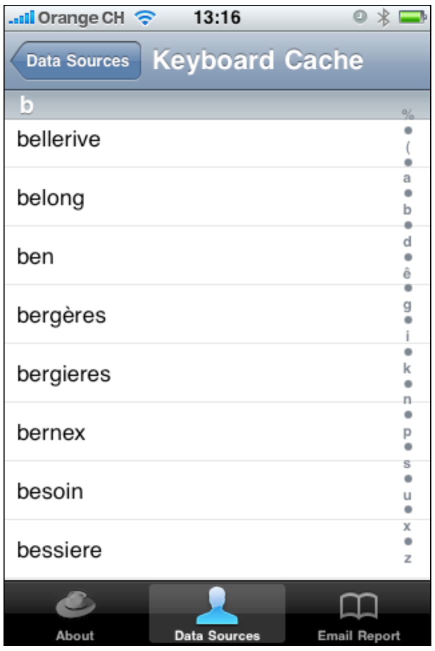
[redacted]@smardtv.com

[redacted]@nagra.com

[redacted]@smardtv.com

[redacted]@wanadoo.fr

Figure 7: AddressBook emails



Keyboard Cache

b

bellerive

belong

ben

bergères

bergieres

bernex

besoin

bessiere

Figure 8: The keyboard cache

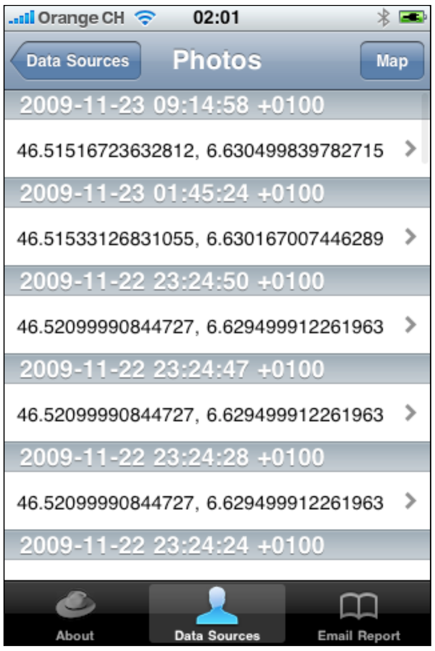
```
# hexdump -C dynamic-text.dat
00000000  44 79 6e 61 6d 69 63 44  69 63 74 69 6f 6e 61 72  |DynamicDictionar|
00000010  79 2d 34 00 00 00 00 02  63 6f 6e 66 65 72 65 6e  |y-4.....conferen|
00000020  63 65 00 74 65 73 74 00  00 00 00 00 00 00 00 06  |ce.test.........|
00000030  01 62 6c 61 63 6b 68 61  74 00 01 68 65 69 67 00  |.blackhat..heig.|
00000040  01 76 64 00 01 67 76 61  00 01 6c 68 72 00 01 69  |.vd..gva..lhr..i|
00000050  61 64 00                                          |ad.|
00000053
```

Figure 9: A keyboard cache file. There is one per language.

For instance, our spyware knows that, on January 19th, I was as at school at 19:28 and then at Café du Simplon at 20:18 (figure 13).

4.4 Valuable data

Safari recent searches (figure 3), YouTube history (figure 4) and your keyboard cache (figure 8) give clues about your current interests. These interests are linked with your name and your email addresses (figure 6), your phone number (figure 5) and your area (figure 12). Harvested from large numbers of users, such data have a huge value in the underground market of personal data, and it must be assumed that trojans are in fact exploiting this on the App Store.


Orange CH 02:01

Data Sources | Photos | Map

2009-11-23 09:14:58 +0100

46.51516723632812, 6.630499839782715

2009-11-23 01:45:24 +0100

46.51533126831055, 6.630167007446289

2009-11-22 23:24:50 +0100

46.52099990844727, 6.629499912261963

2009-11-22 23:24:47 +0100

46.52099990844727, 6.629499912261963

2009-11-22 23:24:28 +0100

46.52099990844727, 6.629499912261963

2009-11-22 23:24:24 +0100

About | Data Sources | Email Report

Figure 10: Geotagged photos. . .

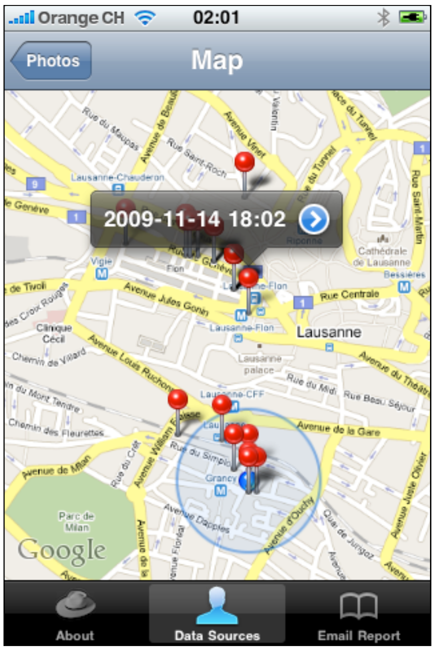
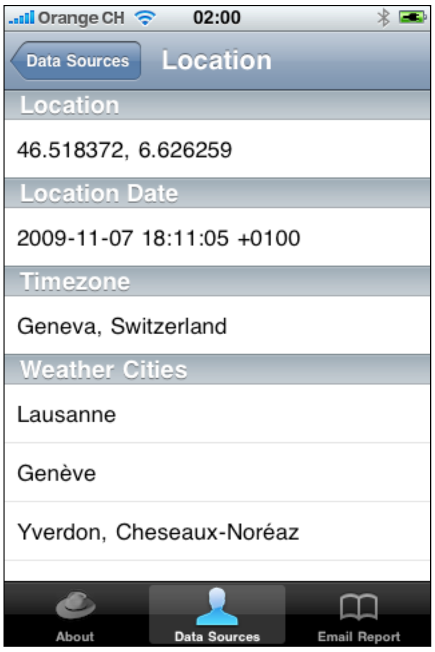


Figure 11: . . . displayed on a map


Orange CH 02:00

Data Sources | Location

Location

46.518372, 6.626259

Location Date

2009-11-07 18:11:05 +0100

Timezone

Geneva, Switzerland

Weather Cities

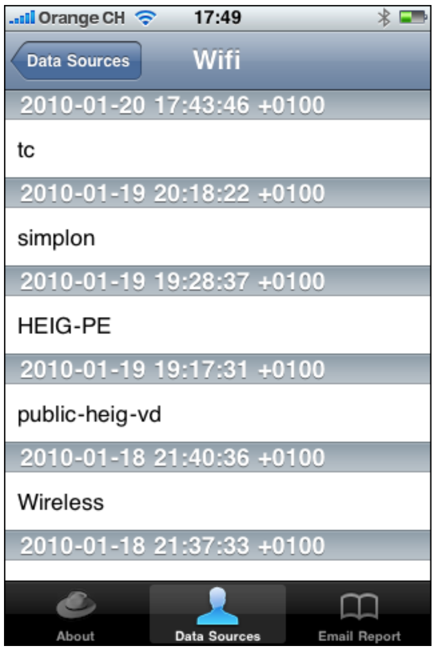
Lausanne

Genève

Yverdon, Cheseaux-Noréaz

About | Data Sources | Email Report

Figure 12: Location guess


Orange CH 17:49

Data Sources | Wifi

2010-01-20 17:43:46 +0100

tc

2010-01-19 20:18:22 +0100

simplon

2010-01-19 19:28:37 +0100

HEIG-PE

2010-01-19 19:17:31 +0100

public-heig-vd

2010-01-18 21:40:36 +0100

Wireless

2010-01-18 21:37:33 +0100

About | Data Sources | Email Report

Figure 13: Wifi connection log

Also, the approximate location (figure 12), recent calls (figure 5, geo-tagged photos (figure 10) and Wifi connection logs are potentially valuable for forensics purposes.

5 Getting published on the App Store

To be published on Apple's App Store, an application must be submitted by a developer enrolled in the (paid) "iPhone Developer Program"[24]. Apple only gets the executable file, not the source code. Next, the application goes through an approval process which mainly looks for user interface inconsistencies but also undocumented function calls and yes, malware. The challenge for a malware author is to go unnoticed through this process. When approved, the binary is properly signed by Apple and can be run on any device. For paid applications, Apple keeps 30% of the revenues.

5.1 App Store and malware

5.1.1 The purpose of reviews

Apple's tight control over the App Store is resented by many[25] developers[26]. Moreover, it normally takes about one week, which may be prohibitively slow for security updates. This delay can be shortened by asking for an expedite review, though. The reasons for rejection are often[27] perceived to be arbitrary and unreasonable.

Apple's senior vice president Phil Schiller explained[28] that Apple needs to ensure that everything works as expected by consumers. Apple also intends to reject applications which are in a legal gray zone such a casino gambling or collecting personal data, which seems only reasonable. Nonetheless, with some 10,000 binaries submitted each week, invariably, some malware are going to sneak through.

5.1.2 iPhone software development kit agreement

Every iPhone developer is bound by the terms of Apple's iPhone SDK Agreement. Unfortunately, this agreement forbids iPhone developers from commenting on its terms and conditions:

[24] http://developer.apple.com/iphone/program/

[25] http://www.joehewitt.com/post/innocent-until-proven-guilty/

[26] http://www.paulgraham.com/apple.html

[27] http://www.tuaw.com/2008/08/07/thoughts-on-the-iphone-app-store-review-process/

[28] http://www.businessweek.com/print/technology/content/nov2009/tc20091120_354597.htm

5.4 You may not issue any press releases or make any other public statements regarding this Agreement, its terms and conditions, or the relationship of the parties without Apple's express prior written approval, which may be withheld at Apple's discretion.

As a consequence, I won't be commenting further here, except to note that it states that developers should comply with local laws and regulations, and should not harvest users' personal information. Spyware submitted to the App Store are rejected on this basis.

The content of the agreement has been divulged on Wikileak and Wired[29].

5.2 Hiding the beast

5.2.1 The review process

The exact App Store analysis reviewing process is unknown and the following considerations are only guesswork based on actual App Store rejections[30] and knowledge of Apple development tools. Nonetheless, it is safe to assume that Apple probably mixes static and dynamic analysis.

Note that even if Apple had access to the source code, it probably could not afford a full security code review, and malware authors would just hide malicious behavior a bit better, as demonstrated each year in the *Underhanded C Contest*[31].

5.2.2 Static analysis

The static analysis probably consist of dumping the strings in the binary file — say with the "`strings`" command — and checking them against a black list of forbidden class and method names as well as file paths. This process can easily be circumvented with simple yet effective obfuscation techniques, as shown in listing 3.

Listing 3: Simple yet effective string obfuscation technique

```
- (NSString *)stringMinus1:(NSString *)s {
    NSMutableString *s2 = [NSMutableString string];
    for(int i = 0; i < [s length]; i++) {
        unichar c = [s characterAtIndex:i];
        [s2 appendFormat:@"%C", c-1];
    }
    return s2;
```

[29] http://wired.com/images_blogs/gadgetlab/files/iphone-sdk-agreement.pdf

[30] http://appreview.tumblr.com/

[31] http://underhanded.xcott.com/

```
}

- (void)viewDidAppear:(BOOL)animated {
    NSString *pathPlus1 =
        @"0wbs0npcjmf0Mjcsbsz0Qsfgfsfodft0dpn/bqqmf/bddpvoutfuujoht/qmjtu";
     // @"/var/mobile/Library/Preferences/com.apple.accountsettings.plist"
    NSString *path = [self stringMinus1:pathPlus1];
    NSDictionary *d = [NSDictionary dictionaryWithContentsOfFile:path];
    // ...
}
```

5.2.3 Dynamic analysis

The dynamic analysis probably consists in running the “I/O Activity” instrument in Apple’s Instruments application and checking the paths of opened files against a black list of forbidden paths. Such a process is easily circumvented by delaying the malicious behavior to a later date, triggered by a flag to be set remotely or activated only in a specific geographic area.

5.2.4 Private APIs

Apple is known to look for calls to private APIs (undocumented APIs) and reject applications on that basis. This is why our SpyPhone application does not use any of them and indeed it does not really need to.

However, a real spyware author probably wouldn’t mind and would take advantage of Objective-C’s ability to lookup classes and selectors by name at runtime. This way, he could define them remotely or obfuscate their usage.

According to my research, using private APIs doesn’t help much in harvesting personal data. The most interesting piece of data is the IMEI number (listing 4), which uniquely identifies a specific mobile phone being used on a mobile network.

Private APIs can be browsed with iPhone Objective-C Runtime Browser[32].

Listing 4: Using a private API to dynamically retrieve the IMEI number

```
    NSString *path = @"/System/Library/PrivateFrameworks/Message.framework";
    BOOL bundleLoaded = [[NSBundle bundleWithPath:path] load];

    Class NetworkController = NSClassFromString(@"NetworkController");
    NSString *IMEI = [[NetworkController sharedInstance] IMEI];
```

[32]http://runtimebrowser.googlecode.com

5.3 Apple's kill switch

Rumor has it that Apple can remotely deactivate iPhone applications. In July 2008, iPhone forensics expert Jonathan Zdziarski [3] discovered a URL (see figure 14) with a Core Location[33] black list. This black list can prevent an application from using the Core Location functionalities, as an anonymous contact at Apple confirmed[34] to noted Mac blogger John Gruber. However, Apple has apparently never used the black list nor acknowledged its existence publicly.

How would our SpyPhone be affected by this kill switch? The short answer is: not at all! SpyPhone uses various informations to estimate the location, including the last GPS lookup from Maps, and does not use Core Location directly.

```
# curl https://iphone-services.apple.com/clbl/unauthorizedApps
{
        "Date Generated" = "2010-01-20 17:23:37 Etc/GMT";
        "BlackListedApps" = {};
}
```

Figure 14: Core Location black list

6 Attack scenarios

Here are some attack scenarios, outlining the potential consequences of a "privacy attack" and illustrating ways in which iPhone security is not as good as it should be.

6.1 The spammer

A breakout game is made available for free on Apple's App Store. While you are playing breakout, it reads your email address, your recent Safari searches, your weather cities and the words contained in your keyboard cache.

When you submit your high score to the application's server, stolen information is sent at the same time in an encrypted form. The application also sends all the email addresses in your address book.

[33]Core Location is the iPhone's framework for locating the device's position through GPS or A-GPS.
[34]http://daringfireball.net/2008/08/core_location_blacklist

Now the spammer knows your interests from Safari searches and the keyboard cache. It also knows your location thanks to weather cities. This information can then be used to send you targeted commercial offers, or it can be sold on to other parties.

6.2 The blackmailer

A collaborative application on Hollywood gossip is made available for free on the App Store. While giving clues about spotting stars, it surreptitiously goes through your address book and edits the email addresses.

Knowing that film industry people are likely to download this application, the emails they send are diverted to a clandestine server, providing potentially compromising private information to a prospective blackmailer.

The approach can be tailored to produce the same scenario in the industrial, political or financial world.

6.3 The luxury products thief

An application for Rolls Royce owners or art collectors could report the name, the area, the phone and the geotagged photos of wealthy people. This is enough informations to rob them, especially if it can be determined that the targeted individuals are currently away from home.

6.4 The jealous husband

Unlike the previous scenarios, this one needs a physical access to the device.

A detective, an evil competitor or even a jealous husband may be interested in stealing the personal data in an iPhone to which they have physical access. All that is needed to do so is a Mac, a 99 USD Apple developer license and a USB cable. It takes just five minutes to install SpyPhone, steal the personal data with the "email report" function, erase the evidence by deleting the sent mail and delete SpyPhone itself.

Nothing has been jailbroken, no expensive device wiper was involved, and a horde of valuable personal data has been stolen. The iPhone owner need never suspect anything, and the jealous spouse or business competitor has full liberty to study Wifi connection logs, photo library dates and geotags for evidence that, say, the partner was not at the office at a given time.

Figure 15: French Prime Minister and French former Justice Minister using an iPhone (businessweek.com).

Figure 16: François Fillon showing his iPhone on TV (iphon.fr).

6.5 VIPs

It is easy to imagine how an attack could be targeted against a particular individual. For example, French Prime Minister François Fillon (figures 15 and 16) is very proud of his iPhone and takes it everywhere. Fillon is a native of the French region called *la Sarthe*, where he also has his political roots. There is a significant likelihood that he would download an iPhone application designed to provide local breaking political news. It does not take much imagination to see the potential for damage in such a scenario.

7 Recommendations

7.1 For Apple

7.1.1 Don't rely on security through obscurity

First of all, Apple should stop claiming [1] that an application cannot access data from other applications (see 4.2.3). This is clearly wrong, misleading and dangerous. Secondly, Apple should decide if the observed behavior, present since day one, is a flaw or not. If it is not, then Apple should document it properly. But if it is, Apple may have to review its secure software development lifecycle (S-SDLC) process. Given Apple's interest in corporate environments and the level of competition, this is a matter of credibility.

7.1.2 Wifi connection log and keyboard cache

There is no reason why the wifi connection logs should be readable. The same applies to the keyboard cache, which should be an OS service associated with text fields. It should not be possible to retrieve their whole contents.

7.1.3 Firewall and analytics frameworks

The iPhone clearly lacks an optional outbound firewall similar to LittleSnitch[35] on the Mac. Such a service would allow people to opt-out from the various analytics frameworks. An alternative would be to implement a single switch that gives consumers some kind of universal or per framework tracking opt-out option. This is what PrivaCy[36] does in the jailbroken world.

At the end of the day, the fact that so many developers use analytics frameworks may be a hint to Apple to give more usage information to developers, and provide users a setting to opt-out.

7.1.4 Address Book

Users should be required to grant access to the Address Book (see 4.2.2) individually for each application, as is currently the case for the Core Location framework. A breakout game has no business accessing your contacts. Moreover, a separate request should be prompted before editing the Address Book. I may want to authorize a quick dialer to retrieve my contacts, but I certainly don't want it to change them without my knowledge and consent.

The risk, naturally, is that the security rules and settings become overwhelming, as in Windows Vista or in the Java security model, which shifts the burden entirely to the consumer. The resulting annoyance and irritation leads many users to simply allow everything.

7.1.5 Towards Apple approved security policies?

To stay in line with the current model where most of the iPhone security depends on Apple and not the user, Apple could ask application developers to establish a security policy for their applications. For instance, the `Info.plist` file could contain a Policy dictionary allowing HTTP access to a specific URL and Core Location usage, but preventing Address Book or file system access. These policies would then be assessed by Apple's reviewers (figure 17). This might be one way to mitigate risk without the user having to manage complex security policies.

This approach is actually a very simple version of a self-defined sandboxing principle called model-carrying [7]. The generation of models could be automated, and it would be possible for the App Store pages to describe it

[35] http://www.obdev.at/products/littlesnitch/

[36] http://cydia.saurik.com/package/com.saurik.privacy

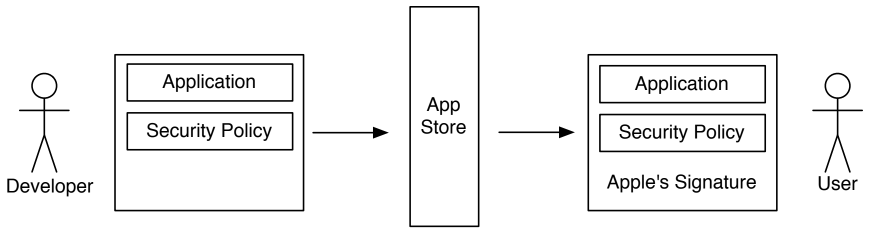


Figure 17: Developers could submit a security policy with their application. Apple would ensure that authorizations are tight enough, and risks would be mitigated without users being overwhelmed with security pop-ups.

in simple terms to potential customers. This is where a closed system could actually be a benefit for users who just want their phone to work.

7.1.6 Device unique identifier

The device unique identifier is currently available through the official SDK's API. While it is not personal data since it cannot identify a physical person, it may be used to aggregate data collected from various applications and analytics frameworks.

As a general rule, an application should not be able to know which other applications you have run. Users merely accept cross-site cookies on the web, they probably would not accept their computer's unique identifier to be transmitted to Google Analytics.

A possible solution for Apple would be to add something like an *app-device identifier* or ADID. This value would be unique for a given device and the requesting application. The current device unique identifier could be kept, but the user should be asked to allow or deny its usage, as with Core Location.

7.2 For consumers

Consumers should be aware that iPhone security is far from perfect and that a piece of software downloaded from the App Store may still be harmful. As a basic precaution, users should regularly clean the browser's recent searches and the keyboard cache in Settings. They should also change or delete the declared phone number, also in Settings.

7.3 For professionals

Professional users should avoid running untrusted applications, especially if they are required by law to protect data confidentiality. This includes groups such as bankers, attorneys, medical staff, law enforcement officers and so on. Also, legal departments should be aware that confidential data may already have leaked.

An answer to this potential spyware issue may be Apple's program for iPhone enterprise deployment[37], which lets administrators create configuration profiles enforcing restrictions such as disabling Safari or disabling the App Store.

8 Conclusion

Overall, Apple has addressed iPhone security in a sensible manner to protect most of the data. The file system permissions rely both on Unix permissions and on the sandboxing system. Passwords are kept safe, encrypted in the Keychain. Application databases are not readable by other applications. Overall security improves with each new iPhone version, but some basic security principles are still not correctly implemented, in particular the least privilege principle, the deny by default principle and also a recognition that it is dangerous for Apple to deny the existence of vulnerabilities.

Unix permissions, sandboxing system rule sets and App Store reviews are all very well in theory but their actual implementation is flawed. Numerous files are still readable directly by an application downloaded from App Store (section 4.3), mainly preference files, but also the photo library including geotags, the keyboard cache and the Wifi joined networks history. Combined with full access to the Address Book and the phone number, these data are too tempting for cybercriminals not to be exploited by spyware, especially in a store dominated by free and "one-buck" applications.

To be useful, a trojan would still need to be distributed on the App Store. The fact that our SpyPhone application does not use private APIs would greatly help it pass unnoticed through the App Store review. The huge quantity of submissions, and the fact that reviewers only have the binary code to review, make it easy to hide the malicious behavior, as shown in section 5.2. Consequently, it must be assumed that spyware are currently on the App Store.

[37]http://www.apple.com/iphone/business/integration/

While the fact that personal data are accessible will not be a complete surprise for experienced iPhone developers, the scale of such access probably will. Some may consider that this is unavoidable and therefore normal. Such an attitude, however, amounts to denial, and exposes users to major privacy incidents including massive identity theft and other targeted attacks (section 6).

It is a matter of concern that, two years and a half after iPhone's introduction, and despite its huge commercial success, Apple has not fully addressed several basic filesystem privacy issues and, even worse, continues to disseminate misleading information in its public documentation on iPhone security (section 4.2.3). The danger is therefore that the problem, far from being a bug that you can fix, may be closer to something like a design flaw.

I hope that this paper will give the general public and professionals a clearer view of the real risks, and some clues on how to assess and mitigate these risks, so that they can decide themselves how they should use their iPhones properly in their private lives or corporate environments.

I also want to make it clear that you should have no illusion: in the fast-paced environment of smartphone operating systems, every platform has its own flaws. To Apple's credit, it has at least a first line of defense against harmful applications, in the form of App Store reviews. At a time when malware are appearing on Android Marketplace[38], this defense should be retained and improved.

Acknowledgments

I would like to thank everyone who has taken the time to review this paper and exchange their views with mine, especially Graham Lee (@iamleeg), Dominique Bongard (@reversity), Eduard Friesen and Joël Spaltenstein, as well as several other anonymous reviewers.

References

[1] Apple, *iPhone in Business: Security Overview*, http://images.apple.com/iphone/business/docs/iPhone_Security_Overview.pdf

[2] Swiss Federal Data Protection and Information Commissioner, *La protection des données et Internet*, http://www.edoeb.admin.ch/dokumentation/00898/

[38] http://www.theregister.co.uk/2010/01/11/android_phishing_app/

[3] Jonathan Zdziarski, *iPhone Forensics: Recovering Evidence, Personal Data and Corporate Assets*, O'Reilly Media, September 2008, http://oreilly.com/catalog/9780596153595

[4] Sharon D. Nelson and John W. Simek, *Why Lawyers Shouldn't Use The iPhone: A Security Nightmare*, Sensei Enterprises, Inc http://www.senseient.com/articles/pdf/iphone_security.pdf

[5] Phillip Porras, Hassen Saidi and Vinod Yegneswaran, *An analysis of the Ikee.B (Duh) iPhone botnet*, SRI International, Menlo Park, USA http://mtc.sri.com/iPhone/

[6] Andrew Hoog and Kyle Gaffaney, *iPhone Forensics*, viaForensics, http://viaforensics.com/wpinstall/wp-content/uploads/2009/03/iPhone-Forensics-2009.pdf

[7] R. Sekar and V. Venkatakrishnan and S. Basu and S. Bhatkar and D. DuVarney, *Model-carrying code: A practical approach for safe execution of untrusted applications.*, In ACM Symposium on Operating System Principles (SOSP), Bolton Landing, New York, October 2003. http://www.cs.sunysb.edu/~sas/papers/nspw01.pdf

A Sandboxing rules

Here is the file /usr/share/sandbox/SandboxTemplate.sb, which documents the rules enforced at the kernel level. Comments are part of the original file.

```
(version 1)
(deny default)

; Sandbox violations get logged to syslog via kernel logging.
(debug deny)

(allow sysctl-read)

; Mount / umount commands
(deny file-write-mount file-write-umount)

; System is read only
(allow file-read*)
(deny  file-write*)

; NOTE: Later rules override earlier rules.

; Private areas
(deny        file-write*
       (regex "^/private/var/mobile/Applications/.*$"))
```

```
(deny        file-read*
        (regex "^/private/var/mobile/Applications/.*$"))

; SQLite uses /private/var/tmp
; TBR: <rdar://problem/5805879> SQLite doesn't honor
; the TMPDIR environment variable
(allow        file-write*
        (regex "^/private/var/tmp(/|$)"))
(allow        file-read*
        (regex "^/private/var/tmp(/|$)"))

; TBR: <rdar://problem/5806524>
(allow process-exec
        (regex "^/private/var/tmp$"))

; TBR: <rdar://problem/5830139>
(allow file-write*
        (regex "^/private/var/tmp/UpdatedSnapshots/$"))

; Permit reading and writing in the App container
(allow        file-read*
        (regex "^/private/var/mobile/Applications/ \
        XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX(/|$)"))

(allow        file-write*
        (regex "^/private/var/mobile/Applications/ \
        XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/(tmp|Library|Documents)(/|$)"))

(allow        process-exec
        (regex #"^/private/var/mobile/Applications/ \
        XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/.*\.app(/|$)"))

; Allow Address book access via filesystem
; This is an SQLite3 database - there is room to make the rules tighter
(allow        file-write*
        (regex "^/private/var/mobile/Library/AddressBook(/|$)"))
(allow        file-read*
        (regex "^/private/var/mobile/Library/AddressBook(/|$)"))

; Allow keyboard db access via filesystem
; This is a custom file format.  There is room to make the rules tighter
(allow        file-write*
        (regex "^/private/var/mobile/Library(/Keyboard)?(/|$)"))
(allow        file-read*
        (regex "^/private/var/mobile/Library(/Keyboard)?(/|$)"))

; Pictures, but not other media
; Allow photo access via filesystem. There is room to make the rules tighter
(deny        file-write*
        (regex "^/private/var/mobile/Media(/|$)"))
(deny        file-read*
        (regex "^/private/var/mobile/Media/"))

(allow  file-write*
        (regex "^/private/var/mobile/Media/com.apple.itunes.lock_sync$"))
(allow  file-read*
        (regex "^/private/var/mobile/Media/com.apple.itunes.lock_sync$"))

(allow        file-write*
        (regex "^/private/var/mobile/Media/DCIM(/|$)"))
(allow        file-read*
        (regex "^/private/var/mobile/Media/DCIM(/|$)"))

(allow  file-read*
        (regex "^/private/var/mobile/Media/Photos(/|$)"))

; Mach lookups.  There is room to make the rule tighter.
(allow mach-lookup)
;;       (global-name "PurpleSystemEventPort")
;;       (global-name "com.apple.CARenderServer")
;;       (global-name "com.apple.eventpump")
;;       (global-name "com.apple.springboard.migserver")
;;       (global-name "com.apple.system.notification_center"))

(deny process-fork)

; For ASL logs - /var/run/asl_input (XXX: socket can now be named)
; (allow network-outbound)
;        (to unix-socket "/private/var/run/asl_input"))

(allow network*)

; To allow crash reporter / exceptions to kill the process
(allow signal (target self))
```