Bachelor thesis
Information Science

Radboud University

Smartphone privacy demo

Author:
Michael Jansen
4196252

First supervisor/assessor:
Dr. Peter Schwabe
p.schwabe@cs.ru.nl

Second assessor:
Erik Poll
erik@cs.ru.nl

June 23, 2015

Abstract

Smartphones are very popular today. Nearly everyone in our area is using one and is especially using WiFi to connect to the internet. This connection is normally established automatically, so that the user does not have to enter the credentials every time. To find known networks, smartphones are searching for them the whole time. While searching, they offer private data that can be captured by anyone. We try to do that: collect the data, analyze and match it with each other and computer new information that was not presented originally.

Since the common smartphone user is not aware of privacy risks and most of the time chooses for comfort above the security of his own data, we want to make him aware of how much information one can gather and compute about him without his participation. Since those people quickly get bored when trying to *explain* it, we want to *show* it. Therefore, we display the collected and computed data about them (as soon as there is enough) at the moment they are present so that a screen can "welcome" them with an overview of things we know about them that they didn't expect.

Contents

List of Figures

Chapter 1

Introduction

1.1 Smartphones and WiFi

Today, more than every fifth person on the world is using a smartphone [7]. Such a smartphone is a combination of a telephone, an Internet device, a music player and a lot of other functionality that can be added through "apps" – small applications that run on the smartphone.

Nearly every functionality of the device uses the Internet, e.g. surfing, checking e-mails, listening to Internet radio stations, checking Facebook, Twitter, etc., looking up directions and many other scenarios. Since it is way more expensive for Internet Service Providers (ISPs) to maintain their radio towers than to provide Internet via cable or DSL, it is also more expensive for customers to use their mobile data connection than using WiFi, no matter whether at home, at the work place, university, with friends or in a public café. Therefore, nearly every smartphone user tries to use WiFi networks as often as possible. For their comfort, the smartphones try to automatically reconnect with a known WiFi network as soon as it is in range. The smartphones scan the environment for known hotspots. While searching, the phones need to send a lot of private information in order to find those hotspots. This data can be captured by anyone.

1.2 Motivation

Sadly, only few people are aware of this. When trying to explain this to people that are not interested in information and communication technology, they quickly get bored, so a new approach would be helpful. Specifically, we will take these three steps:

1. We will install a modified WiFi access point that just captures the searching of the devices. Personal data will be gathered on the *router*.

2. We will analyze the data and create a profile for each phone. These

profiles will also be linked and prepared for displaying them. This will be done on the *server*.

3. Thereafter, we will install a *screen* in the surroundings of the hotspot. As the router detects a phone, it sends its appearance to the server which will then present the corresponding profile.

We think that the data, collected over a few weeks and linked with each other, will also deliver new information and insights. For example, social relations can be discovered by searching for smartphones that are detected always at the same time.

Further, our contribution lies in the third point: with the demo, which is ultimately a continuous presentation of our collected and computed data, we will warn people and show also those who are not aware of cyber crime what can be done with their data.

Chapter 2

Preliminaries

2.1 MAC addresses

Today, every modern device is connected with others. This happens via networks that consist of different layers, implemented in the TCP/IP protcoll. If a device wants to talk with another, it has to go through these layers down to the physical layer where the transmission actually takes place. On top of that layer, there is the Data Link Layer which is needed to *identify* the right receiver. This identification takes place based on its Media Access Control (MAC) address, so it is a unique identifier to find the right network interface that is attached to the receiver machine.

A WiFi adapter is also such a network interface, which means that every device can uniquely be identified by its MAC address.

2.2 Service discovery

Smartphones normally try to reconnect to known networks as soon as they are in range. In order to do so, the access point that is connected to the Internet and the smartphone have to discover each other. This *service discovery* happens in two different ways (*passive* and *active* refers to the point of view of the device that wants to *get* an Internet connection):

Passive: The access point (AP) sends its own identitiy regularly. The packets it sends are called *beacons* that can be seen by the *clients* which are searching for networks. If it is one of the networks the client knows, it connects to it.

Active: The client actively searches for networks it knows. To find them, it sends the name of all the networks in so-called *probe requests*. Each of them contains the name of one known network. If an AP receives such a probe requests that was searching for it, it answers with a *probe response* that is used by the searcher to connect to it.

2.3 Receiving probe requests

Our goal is to show privacy leakage. We are using probe requests to achieve this. To capture them, we reconfigured a standard home network router. Since we do not want the clients to connect with our AP but do want to read the packets that go through the air, we put the WiFi adapter in *monitor mode*. It then becomes a passive antenna that receives every packet on the frequency it is adjusted to. We change that frequency regularly to catch as many requests as possible. There are defined frequencies that represent *channels*. The repeated changing of the frequency is called *channel hopping*.

2.4 Flashing a router

Linksys is one of the major router manufacturers. To run their devices, they need an operating system. In 2003, it turned out that Linksys used Linux with software that was published under the GPL license [9]. This license grants the right to use the software that was published under it without charge. In exchange, the software that uses the GPL licensed software has to be made open source, too. Linksys did not take this into account, used GPL software on its routers and sold them. When it turned out that they did so, they had to publish their source code. That gave developers the chance to write an open operating systems that would run on the chipsets Linksys used in its routers. One of these new operating systems is OpenWRT [10]. Once installed on the router (this process is called *flashing*), it is accessible via a command-line interface and a web interface and it is also possible to install software on it.

2.5 HTTP requests and responses

Since we want to present information to the user, we chose to do this via a website that runs in a browser on full-screen. To communicate with a web server, that website has to make HTTP requests. Such a request can be of different kinds. We will use those two:

GET In order to *receive* data from the server, the GET request is mainly used. It can contain some parameters but according to the HTTP standard, it should not cause any changes made on the server, e.g. alter a database. The URL is used to transfer these parameters, e.g., `http://example.org?param1=value1`.

POST If the sender want to transmit a lot of data to the server, a POST request is advised. Therefore, the request body is used instead of the URL to send the data. The server then can process the data sent within the request body.

For every HTTP request, the server answers with an HTTP response. That does not only contain a status code (e.g. `200 OK`, `404 Not Found`, `500 Internal Server Error`, ...), but also some content. If a user just wants to visit a website, the browser sends a GET request to the URL the user typed in. The content of the HTTP response is the HTML page shown to the user.

2.6 AJAX

AJAX means Asynchronous JavaScript and XML. Traditionally, for each step or at every time that new data has to be sent to or received from the server, the webbrowser sends a request to the server, eventually containing some new data, and gets a response containing a whole HTML webpage that then is shown to the user. AJAX creates a new layer: instead of the core of the webbrowser, the JavaScript engine (that is part of the webbrowser) makes the HTTP request to the server and gets the answer. That does not necessarily have to be a HTML webpage but could, for instance, also be an XML document (that's where the name comes from). The JavaScript application then can process that XML document and alter the elements on the webpage instead of loading a whole new page where possibly only a part has changed. This saves loading time and creates a browsing experience that is more seamless.

Chapter 3

Setup

As shown in Figure 3.1, the setup of our system consists of three parts: router, server and client. The router receives all the probe requests sent by devices nearby and sends them to the server. On top of that, it sends information about which devices are currently in range. The server receives and processes the new information by analyzing and storing them into the right tables. Also, it regularly updates the profiles of the known devices and thus generates information based on the data it has stored. The client asks the server continuously which profiles it should display. As soon as it receives new profiles, they appear on the screen. If the server does not mention those profiles anymore, they disappear.

3.1 Router

The router has two tasks: it captures the packets that are sent through the air and it forwards them to the server.

We are using a Western Digital My Net N600, a standard home network router which we flashed with OpenWRT. Through this, we were able to install *aircrack-ng* [1], an application suite that was initially designed to crack password-protected networks. In our case, we are using it to put the first WiFi adapter (2.4 GHz) into monitor mode using *airmon-ng*, one part of the suite, by running the following command:

```
airmon-ng start wlan0
```

Airmon-ng creates a new interface called `mon0` that can be treated as a WiFi interface in monitor mode. We are then using *tcpdump* [11] to receive WiFi packets that are sent through the air. Communication with the server is done via HTTP requests using *cURL* [5], a command line tool to transfer data via a network. We are using HTTP POST requests which have the output of tcpdump as body. Every line represents one probe request. What such a probe request looks like is explained in Section 4.1.1.

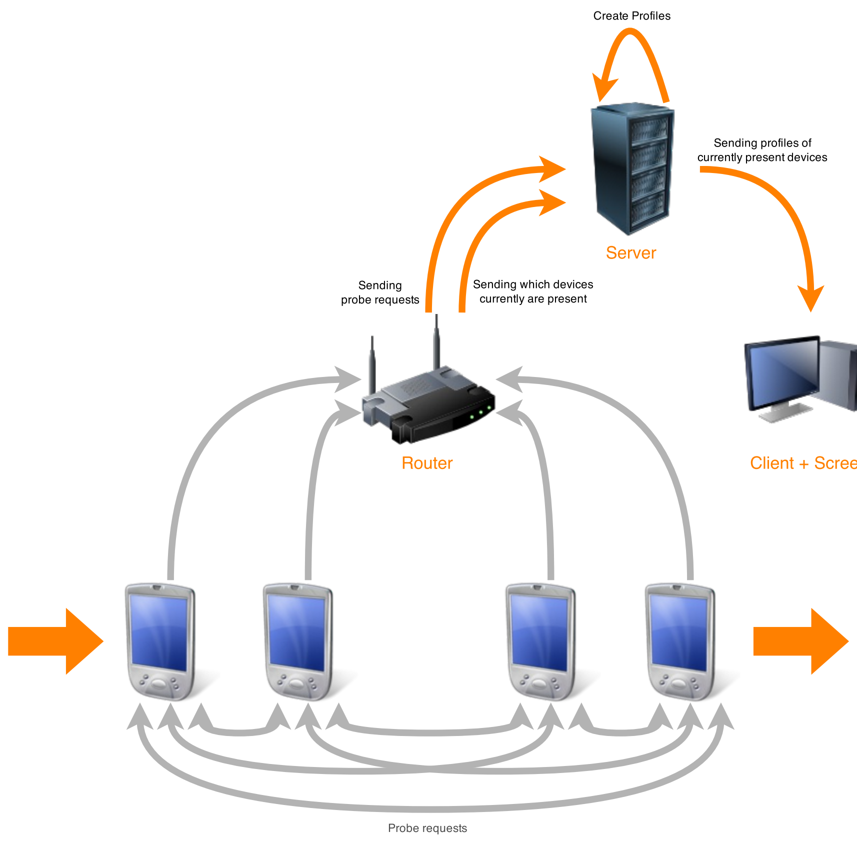
Create Profiles
Sending profiles of currently present devices
Server
Sending probe requests
Sending which devices currently are present
Router
Client + Screen
Probe requests

Figure 3.1: Experiment setup

3.2 Server

The server receives raw data about captured requests and has to store them structurally. It also has to update profiles and compute new information about devices regularly in the background. When asked by the client, the server has to present the data about currently active devices that the router informed the server about.

As server, we are using a computer with an Intel Celeron G1820 processor at 2.7 GHz (x2), a 500 GB HDD and 2 GB of RAM. On that system, we run Ubuntu 15.04 64 bit as operating system and installed a standard Apache webserver with PHP to process data and MySQL to store it structurally. In Figure 3.2, our database structure is visualized. We will now explain each table and its task.

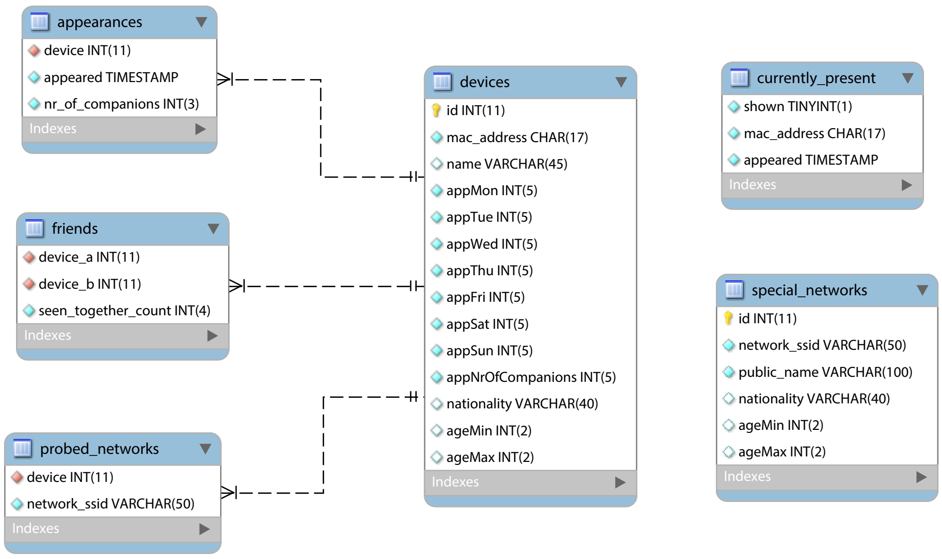

Figure 3.2: Database structure

`devices` stores all the devices, giving them a unique number that is associated with the MAC address of a device. We also reserved a column for the name of the device, but we cannot establish the name automatically. It will give us the possibility to personalize the demo later on, if people are actively willing to be addressed with their name or a pseudonym so that they can easily identify which of the shown profiles is theirs. In addition, we store a count how often the device is seen per day of week, a guessed nationality and guesses about the minimum age and maximum age of the owner. We also store the average number of companions the device was seen with.

`appearances` contains one row for each appearance of a device. It also stores how many devices are seen at the same time.

`probed_networks` are those networks the client is searching for. There are no duplicates, so if a device searches again for the same network, no new row will be created.

`friends` will help us later to discover friendships. Each time two devices are seen together, the counter increases. To prevent duplicate entries (e.g. device_a = 1, device_b = 2 would have the same meaning as device_a = 2, device_b = 1), the devices are ordered according to their MAC address and after that, the row is inserted or updated in case the friendship count already existed.

`special_networks` consists of networks such as "eduroam". Networks in this table will give us more information about the user. Eduroam, for instance, means that the owner of the device is most likely a student or university teacher. Furthermore, we make guesses about how old a user of that network is. Therefore, we store a minimum and a maximum age. If such a guess is not possible (for instance, the public train network does not say anything about the maximum age of the user), we leave the field empty. In addition to that, a guess of nationality is stored for people that want to connect with that special network.

`currently_present` contains the MAC addresses of devices that are currently in the surroundings of the router. Furthermore, the last time it was seen is recorded and whether it was already displayed on the screen.

3.3 Client

In our case, the computer that runs the web server is also the client which displays the information. Nevertheless, we treat it as a separate computer since its work could be done on another machine without any change.

The client is thus connected to the same network as the server, being able to make HTTP requests to the server. This is done on a standard HTML website that contains our JavaScript application. The HTTP GET requests are done via AJAX and the transmitted data is formatted in JSON (JavaScript Object Notation). This website is shown within a standard webbrowser that is running on the client in full-screen mode. The client is connected to a screen visible to the people.

Chapter 4

Running the demo

4.1 Analysis

4.1.1 Data gathering

To create profiles, we need information about the devices. We are capturing probe request to gather this information. After putting the WiFi interface into monitor mode, we run tcpdump with a filter so that only probe requests are caught:

```
tcpdump -i mon0 -e -s 256 type mgt subtype probe-req
```

Every probe request captured by tcpdump has the following layout:

```
22:14:29.849593 6015968459us tsft 1.0 Mb/s 2467 MHz 11g -33dB
signal [bit 29] BSSID:Broadcast DA:Broadcast SA:60:d9:c7:0c:91:b3
(oui Unknown) Probe Request (Jansen) [1.0 2.0 5.5 11.0 Mbit]
```

`22:14:29.849593`	Exact time the packet was captured
`-33dB`	Signal strength
`SA:60:d9:c7:0c:91:b3`	MAC address of the device that sent the request
`Probe Request (Jansen)`	SSID of the network the device was looking for (between parantheses)

Table 4.1: Interesting parts of a sample probe request

We are using only some interesting parts of that request which we point out in Table 4.1. Since there are only some seconds between the capturing and the insertion in the database, we are simply using a current timestamp while inserting the row into the database as the time when the device was seen. Currently, we also do not use the signal strength. This could be used in future research to exclude some devices that have too poor signal. What we actively use are the MAC address to (re-)identify the device and the SSIDs it is looking for.

Everything is sent regularly to the server using an HTTP POST request. The server then processes the data by adding and updating information in the corresponding tables: If a device is unknown, it is inserted in the devices table and gets a new unique ID. Otherwise, the ID is fetched from the database. After that, the appearance for that device is recorded as well as all the networks it sent probe requests for. Finally, every pair of two devices that were seen together are written into the table `friends`. If the pair already exists, the "seen counter" is increased by one.

4.1.2 Processing

The server regularly updates the profiles of the known devices. Therefore, it analyzes the data to make guesses for the following information:

1. When does the person normally show up?
   e.g. mostly on Mondays to Thursdays
2. How many people are mostly with her/him?
   e.g. always seen with at least 5 other people
3. Which are her/his friends?
   e.g. device #1534 and #1938 are very often seen together with her/him
4. Where is the person from?
   e.g. searching for a lot of typical Dutch networks such as "Wifi in de trein", therefore (s)he could be Dutch
5. How old is the person?
   e.g. searching for a lot of disco networks, so (s)he is probably between 18 and 28

To show information about when the person normally shows up (1), we look at the appearances table and count maximal one appearance per day. We thus compute *on which days* the person showed up, not how often and at what time. Then, those days are associated with the day of the week – we thus calculate on which days of the week the person mostly shows up. We will get an absolute count on how many Mondays, Tuesdays, Wednesdays, ... the person was seen.

Per appearance is also saved how many other devices were seen with the person. We compute the average of that number to guess how many people are mostly with her/him (2).

As pointed out under 4.1.1, every time two devices are seen together a counter increases for this pair of devices. We select per device those other devices that are mostly seen with that device and present them as the `friends` (3).

In the table `special networks`, we collect data about networks that give us information about the devices that connected to it. The network with SSID "Cafe de Fuik", for instance, suggests that the user is living in the Netherlands (guessing the person is Dutch), and is between 18 and 30 years old. For every device, all probed networks are searched in the `special networks` table. The information is collected and the top nationality is chosen as where (s)he is from (4) and the average minimum and maximum age is used to calculate how old (s)he is (5).

4.2 Live

Since we want to warn people about their information leakage, the computed data has to be presented at the time the person passes the screen.

4.2.1 Detecting people

We are again using a WiFi adapter in monitor mode to capture packets that devices send through the air, but since we are now only interested in *who* there is, we do not need to filter the packets by type of request. Therefore, we use tcpdump with the following command:

```
tcpdump -i mon0 -e -s 256
```

Now, the device doesn't have to send out *probe requests* to be discovered by our system—as long as it sends anything, we detect it. The data is again sent to the server via a HTTP POST request, but in a shorter interval so that our application can react faster on devices that are now seen.

It thus sends all the captured packets to the server, which extracts all different MAC addresses and stores them in the table `currently_present`. The standard values apply for the other columns: a current timestamp for appeared, so when the device was seen, and the value 0 for shown because it has not been displayed on the screen yet. If the MAC address already existed in the database, only the column `appeared` is altered to the current time.

4.2.2 Showing profiles

As explained in the Setup, we wrote a JavaScript applications that continuously makes HTTP GET requests to the server which then presents the data. An example of such a JSON-formatted response is shown in Listing 1.

The server looks up the devices stored in `currently_present`. Devices that appeared more than k seconds ago will be deleted since we assume that they are not in range anymore. We started with a value of 30 seconds for k. This value depends on the amount of different devices that are passing by and has to be adjusted to the setting in which the system runs.

The rest of the table is fetched and every record's `shown`-attribute is set to 1 since the server will now give an answer containing all those devices and they will appear on the screen. For devices that had an 0 in their column `shown`, the server looks up the whole profile and adds it to the answer (in Listing 1, this would for instance be the device with ID 1523). For devices that had a 1, the server only adds the ID to the answer (in Listing 1 the second element of the array, 1934). This means that the server only looks up and sends the whole profile once; afterwards, it only provides the ID.

The JavaScript application processes those answers. It has an own list of devices that have to be shown to the viewers. Every time a response arrives, this list is updated. For new devices, the whole profile is saved to the list. For known devices (meaning that the full profile is cached and their ID is present in the response of the server), nothing is changed. Devices in the cached list are removed if their ID is not present in the response of the server. After every update of the list, the devices with their cached profiles are shown to the user. That means that, as long as people are in range of the router, their profile is presented on the screen. As soon as they are leaving, their profile disappears.

Listing 1 Example JSON-formatted answer of the server

```
[
    {
       {
           "id": "1",
           "mac": "60:D9:C7:0C:91:B3",
           "name": "Michaels iPhone",
           "appearances": {
               "days": {
                   "Mon": "1",
                   "Tue": "1",
                   "Wed": "1",
                   "Thu": "0",
                   "Fri": "0",
                   "Sat": "0",
                   "Sun": "0"
               },
               "nrOfCompanions": "2"
           },
           "age":  {
               "min": "18",
               "max": "47"
           },
           "nationality": "Dutch",
           "probed":  [
               "ADR Wireless",
               "anumo",
               "Augustus",
               "eduroam",
               "free-hotspot.com",
               "Freewave",
               "FREEWIFI De Fuik",
               "UTGUEST",
               "WLAN-0FEA00"
           ]
       }
    },
    {
        "id" : 1934
    }
]
```

Chapter 5

Lessons learned

While designing and implementing this demo, we discovered some difficulties.

5.1 Router

As explained in Section 2.4, the first router which caused the open source community to develop its own firmware was the Linksys WRT54G [9]. We therefore tried a Linksys WRT54GL but it turned out that it had too little memory to install and run the software we needed.

To capture the probe requests, there are multiple software programs. One possibility is airodump-ng, a part of the aircrack-ng package mentioned in Section 3.1. It has a GUI that shows the devices captured and their probed networks in a table and hops channels automatically. Another program is tcpdump (also mentioned in Section 3.1). We decided to use the second since it gives an output that is easy to parse: every line represents one captured probe request (explained in Section 4.1.1).

In order to capture packets that are sent over the air, the WiFi interface has to be set into monitor mode. Therefore, we use airmon-ng (see Section 3.1) that creates a new, virtual interface `mon0` that used `wlan0` in monitor mode. Our script then starts tcpdump on the new interface. It is essential to stop `mon0` after using it by running `airmon-ng stop mon0` because otherwise, every time we run airmon-ng with wlan0, a *new* interface is created (`mon1`, `mon2`, ...). Since our script still uses `mon0`, the router is likely to hang.

5.2 Detecting

While trying to detect devices that are currently in range, we looked at the probe request in realtime. We discovered that as soon as devices are connected with a network, they are not searching for other networks anymore.

We therefore have to find a spot for our system where the devices that are passing by are unlikely to be connected with a network. In addition, we removed the filter of only finding probe requests when detecting devices, so that we see *every* packet the smartphone sends.

Another issue is timing. There are three timespans that have to be defined:

- The router searches for packets that are sent over the air and sends its output regularly to the server. How often should it send that per minute?
- The client regularly asks the server which devices are currently present. How often should it ask that per minute?
- The devices saved in `currently_present` have the attribute when they are seen the last time. After what amount of time should they be removed?

Chapter 6

Related Work

As explained in Section 2.2, smartphones try to find known networks. As Cunche explains [4], there are two ways for the so-called "service discovery" according to the IEEE 802.11 [8]. On the one hand, there is the *passive service discovery.* The access point regularly sends its own name (the SSID) and waits for devices to answer. Those messages are called *beacons.* On the other hand, there is the *active service discovery.* The clients (smartphones, tablets, computers, ...) searchs actively for known networks by sending the SSIDs of all networks they were previously connected with. These messages are called *probe requests.* If a network that was searched for answers, this answer is called a *probe response.* Both, request and response, can be captured by every WiFi interface. To accomplish that, the interface has to be in *monitor mode*, which means that it passively captures everything that goes through the air. It is also important to mention (which is also pointed out by Cunche) that no physical access to the devices is needed to run this system.

Today, those techniques are already in use to track people. For instance, Fukuzaki et al. [6] showed that it is possible to simulate the flow of pedestrians in a controlled area by capturing probe requests. To achieve this, the researchers installed multiple WiFi access points that – comparable to our system – look which devices are in range. They also send those data to a central server, but from multiple access points. In the end, they were able to see where pedestrians go to in realtime as long as they are in the range of the access points. In contrast to our system, they only use the MAC address to reidentify people. We loook also at the networks the devices are searching for.

It is also doable the other way around: an app on a smartphone can locate itself by searching for probe requests in the surroundings. Therefore, the smartphone antenna self is in monitor mode. This was shown by Chon et al. [3]. Through this, a person is also trackable, but in contrast to Fukuzaki et al., they have to participate actively by using the app and sending the information.

All those researchers used WiFi signals to identify deivces and analyse the users. This is also possible through GSM: Aloul et al [2] did this on a campus: they installed GSM modems to track the users and used text messages over those GSM modems to display the information gathered by the modems. This shows that preventions against our demo (for instance a MAC-address randomizer) aren't effective.

A MAC-address randomizer changes the MAC-address of a devices regularly so that it is not possible to re-identfy a device based on its MAC address. Stil, we could try to re-identify them by fingerprinting: two devices that search for exactly the same networks are likely to be the same device.

Another difference between Cunche et al. and our system is that they identify friends base on those fingerprints. If two devices have a very similar list of known networks, then it is likely that the two owners know each other or are friends. We identify friends based on how many times two devices were seen at the same time.

Chapter 7

Conclusions

You need to fight for your pricacy or you'll lose it.

– Eric Schmidt

As Eric Schmidt said, everyone has to do something in order to keep his privacy. We showed that this does not only include *not* posting private things on social networks or publishing it on blogs so that it can be found by Google, whose chairman Eric Schmidt is. Privacy can also be violated *without* the user doing something stupid. Just by passing by and having the smartphone on, a user leaks information that can be gathered and used by others. The problem is thus fundamental: if we want comfort and let our smartphone do all those handy things like connecting to networks, determining our position and regularly checking mails, we have to accept that anyone can track us. The data itself may be encrypted (for instance, checking mails via an SSL connection), but the fact *that* we send or receive data at a specific time at a specific place already says something about us. It may be an information with very little weight, but ...

Many a little makes a mickle

– Scottisch proverb

Continuously gathering data over time, we showed that we can draw a better picture about someone. For instance, on which days the person shows up, with whom and with how many companions on average. Not encrypted data like the SSIDs of the networks the device is searching for showed us even more information. We would be able to sell our display as an **adaptive advertising panel**. In the movie Minority Report from 2002, Tom Cruise passed by an advertising panel that showed ads that were personalized for him. We could also personalize this screen: for instance, we could show ads of cafés the person visited in the past or show advertisements that fits the person's age or nationality. This is like Google AdWords in real world.

Wait a minute ...

Bibliography

[1] Aircrack-ng. http://www.aircrack-ng.org/ (accessed 2015-06-09). 9, 25

[2] Fadi Ahmed Aloul, Assim Sagahyroon, A. Al-Shami, I. Al-Midfa, and R. Moutassem. Using mobiles for on campus location tracking. In *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '09, pages 231–235, New York, NY, USA, 2009. ACM. 21

[3] Yohan Chon, Suyeon Kim, Seungwoo Lee, Dongwon Kim, Yungeun Kim, and Hojung Cha. Sensing wifi packets in the air: Practicality and implications in urban mobility monitoring. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 189–200, New York, NY, USA, 2014. ACM. 20

[4] Mathieu Cunche. I know your MAC address: targeted tracking of individual using Wi-Fi. *Journal of Computer Virology and Hacking Techniques*, 10(4):219–227, 2014. 20

[5] curl and libcurl. http://curl.haxx.se (accessed 2015-06-09). 9, 25

[6] Yuki Fukuzaki, Masahiro Mochizuki, Kazuya Murao, and Nobuhiko Nishio. A pedestrian flow analysis system using wi-fi packet sensors to a real environment. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, pages 721–730, New York, NY, USA, 2014. ACM. 20

[7] John Heggestuen. One in every 5 people in the world own a smartphone, one in every 17 own a tablet. http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10. 4

[8] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access

Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std. 802.11*, 2012. http://standards.ieee.org/about/get/802/802.11.html (accessed 2015-06-09). 20

[9] Andrew Miklas. Linksys WRT54G and the GPL. https://lkml.org/lkml/2003/6/7/164. 7, 18

[10] OpenWrt. https://openwrt.org (accessed 2015-06-09). 7, 25

[11] TCPDUMP/LIBPCAP public repository. http://tcpdump.org (accessed 2015-06-09). 9, 25

Appendix A

Appendix

A.1 Rebuilding the setup

Any files that have to be copies on the devices can be accessed in this Git repository: https://github.com/msjansen/smartphone_privacy_demo

A.1.1 Router

1. Flash the router with OpenWRT [10]. More about flashing in Section 2.4.
2. Install aircrack-ng [1], tcpdump [11] and curl [5] on the router.
3. Set up a local network on the router.
4. Copy the files of the folder `router` from the Git repository

A.1.2 Server

1. Install a webserver like XAMPP containing Apache, MySQL and PHP.
2. Import the database.sql file from the Git repository in the phpMyAdmin interface.
3. Copy the files of the folder `server` from the Git repository to the webserver root.
4. Download mootools (http://mootools.net) and copy the file into the folder. Choose `core`, `no compatability`, `full`, `compressed` and version `1.5.1`. The filename has to be `mootools-core-1.5.1-full-nocompat-yc.js`.
5. Enter the MySQL user credentials into `Model/DBCon.php` on line 16 (address, username, password, database name).

6. The Server has to have the IP 192.168.1.10 since the scripts running on the router try to connect to the server on that IP.

Now, start a webbrowser on any device that is connected to the router and enter the IP of the server. On the same time, start the scripts on the router by executing them in the shell.