PREVENTING USER AND HARDWARE TRACKING IN MOBILE DEVICES

by

David Robert Stites

B.S., Purdue University, 2007

A thesis submitted to the Graduate Faculty of the University of Colorado at Colorado Springs in partial fulfillment of the requirements for the degree of Master of Science

Department of Computer Science

2012

This thesis for the Master of Science degree by

David Robert Stites

has been approved for the

Department of Computer Science

by

_____

Rory Lewis, Chair

_____

Xiaobo Zhou

_____

Chuan Yue

_____

Mark Wickert

_____

Date

Stites, David Robert (M.S., Computer Science)

Preventing User and Hardware Tracking in Mobile Devices

Thesis directed by Professor Rory Lewis

Mobile devices, such as smartphones or PDAs, have become increasingly popular with consumers and often provide essential functionality in their everyday life. Usually these mobile devices contain a great deal of sensitive information such as addresses, contacts, ingoing/outgoing call logs, SMS messages and, on the latest models, a calendar, emails and potentially the user's current location. A smartphone or mobile device today can be as powerful as a desktop or laptop in some respects and, while the latest models feature a complete OS, for many users these devices are "just phones" so there is an underestimation of the risk connected to mobile device privacy. There is a currently existing privacy problem associated with user and hardware tracking in mobile devices. Users can be tracked without their knowledge and consent and have rich profiles built about them using their hardware interface address regarding their location and preferences. This information can be potentially cross correlated to other existing datasets to build advertising profiles for these users. The mitigation to this problem using a framework to support randomly generated, disposable hardware addresses.

Dedication

I dedicate this thesis work to my tireless parents, Richard and Jane Stites, my sister Amy Stites and my loving wife, Rachel Stites. Without their knowledge, wisdom, guidance and support, I wouldn't have been able to reach my dreams! All of you have been my best friends and cheerleaders - I love you all.

Acknowledgement

I wish to express my sincere thanks to Dr. Rory Lewis for his encouragement throughout this work.

I wish to express my sincere thanks and gratitude to Dr. Chuan Yue for instilling an interest in privacy and security in me.

I would also like to express gratitude to my previous employers, Gene and Jamie Johnston, for believing in the power of education.

I wish to thank Erik Neuenschwander for guiding me through the corporate approval process so that this work could take place.

I would like to express a sincere thanks to Nathan Stern, Michelle Moore, Josh Rose and Ed Hunnicutt for volunteering to help me collect data sets for this work.

TABLE OF CONTENTS

TABLES

Table

FIGURES

Figure

EQUATIONS

Equation

CODE LISTINGS

Code Listing

CHAPTER 1

INTRODUCTION

Mobile devices, especially cell phones, have changed a great deal from their predecessors from the 1990s. Gone are the days of brick-sized phones with a 1-line displays, 10 analog buttons and several kilobytes of memory. In recent years there has been an explosion [Bickford] of powerful mobile computing devices. These new smart phones and tablets, small enough to fit in your pocket or backpack, hold an immense amount of computing power. Information is available at a simple touch or finger flick and many users use these devices to access a plethora of data or services such as email, personal contacts and websites (including financial account portals) and even perform tasks which formerly were normally reserved for a desktop system such as video conferencing, watching movies or listening to music.

These devices are able to access the internet, download additional software from the internet, send and receive email, browse websites and send and receive SMS messages from other users. In addition to these capabilities, many subscribers use their cell phone as a primary method of communication, storing their personal contacts information (which include street addresses, email addresses, phone numbers, *etc.*) as well as photographs they have taken. Many of these devices have a built-in GPS that allows the user to "geo-tag" photographs and use Location Based Services, such as

FourSquare, Twitter and Facebook in addition to a basic mapping and GPS functionality. Also, the majority of the mobile platforms have “wallet” applications which allow users to store a wide variety of information such as credit card numbers, bank accounts, serial numbers, web credentials, social identities and personal information. While most of the these applications encrypt the contents of the wallet, some of them do not and store the information in plaintext.

Devices that run iOS (iPhone, iPad and iPod Touch), Android and Windows Mobile (which represent the majority of the market) present a brand new computing paradigm in terms of availability, user interface and privacy. These devices are being targeted by attackers as never before [Hypponen]. Today, more than 300 kinds of malware - among them worms, Trojan horses and other viruses as well as spyware - have been unleashed against the devices [Hypponen]. In addition to malevolent code that executes on the device, attackers can setup “IMSI catchers”, which is a device that can act as a false cell tower. This device enables Man-in-the-Middle (MITM) attacks and can be used for eavesdropping and interception of cellular traffic. Although desktop systems still remain the most widely targeted platform, as mobile computing becomes more ubiquitous and powerful, the lines between the functionality of a traditional desktop system and a mobile system will become blurred and each of these more than 1 billion devices will gradually enter the virtual battlefield.

Purpose of the Study

Clearly, these new mobile device capabilities, mixed with the fact that users store personal information on the devices, make mobile devices a prime target for attackers and others who would compromise privacy, who may not necessarily be bad actors. There are three different, basic categories of attacks that can be carried out against mobile devices which are:

Confidentiality attacks: Data theft and data harvesting [Seriot]

Integrity attacks: Phone hijacking [Bickford]

Availability attacks: Protocol based denial-of-service (DoS) attacks carried out against mobile devices and the infrastructure that serves them, and battery draining attacks [Hypponen]

These three categories represent a wide spectrum of security issues and demonstrate that there are many different attacks that an attacker could carry out. Any of the aforementioned attacks could range in severity from "low" to "high" which makes these particular vulnerabilities a significant, under-appreciated problem. Attacks or compromises to privacy fall under the category of "Confidentiality attacks."

Additionally, the scope of the problems with mobile computing platforms are not static. With the rise and increasing ubiquity of the mobile computing platform, new and challenging issues are beginning to emerge. One example of these issues involves user privacy. Many smart phone applications rely on the use of the user's location so that they

can provide a useful service. [Yiu, Tan, Liu] Secure transport, computation and storage of this data aside, there are additional privacy challenges with applications accessing and using this data as the gathered information permits the tracking and logging of of individuals' whereabouts.

To address these issues, many smart phone manufacturers have started building in privacy controls to their operating system. For example, in Apple's iOS 6, there are application-specific privacy settings the user can set to allow or disallow access to various sets of the user's private data such as location, contacts, calendars, reminders and photos. The sandboxed user-space application will call an API to request data access permissions and the OS will check, out of process, to see if this application has been previously allowed or disallowed by the user. If the user has disallowed access, no access will be granted to the user-space application. If the user has previously allowed access, full permissions will be granted to the user-space application. If the user has never been prompted to grant or deny access, the OS will present a modal dialog, out of process, prompting the user to decide whether or not an application should be granted access to the user's data.

For example, an application running on prior versions of iOS has access to the unique device ID (UDID). The UDID is a hexadecimal string that uniquely identifies a particular device. This UDID is able to be seen by users but, previously, they had no way to restrict its use or reset it. In [Egele], the authors surveyed more than 1,400 applications available on the Apple App Store and on Cydia and more than 50% of those applications accessed and transmitted the UDID. In iOS 6, there are three new

application programming interfaces (APIs) for providing developers and advertising networks a means for device identification without exposing any private or confidential information about the user and giving the user positive control over these identifiers.

However, while privacy controls are improving, they do not yet cover some unique identifiers that might be used to track users. One class of such identifiers is the hardware addresses of network interfaces, such as the WiFi MAC address. The author believes these identifiers will increasingly be utilized to harvest data as platforms restrict the use of other unique identifiers and, hence, restriction of the availability of this information will become more relevant for protection of consumer privacy .

The author proposes to reduce the privacy vulnerability associated with hardware addresses *via* a generic framework which implements disposable addresses for network interfaces on mobile devices. The ability to track mobile devices based on their hardware is a pernicious problem. It is pernicious because tracking can be performed without the user having to install any application (although applications can stealthily perform the tracking in the background) and the user cannot prevent the tracking because:

1) the user has not given consent to track in the first place so the user does not know the tracking is occurring, and;
2) since the hardware address is tied to the device, the only way to eliminate the tracking is to get a new device, which is obviously not a scalable solution for the user, partially because it is cost prohibitive.

Arrangement of the Thesis & Scope of the Study

The arrangement and scope of the thesis is as follows:

Chapter 2, Previous and related work. This chapter examines work previously performed that is similar to this work in terms of theoretical ideas. In addition, this chapter also explores some previously-researched general privacy related topics.

Chapter 3, Privacy background information. This chapter explores different types of privacy and confidentiality breaches and their severities. This section also examines potential causes of privacy and information leaks.

Chapter 4, The OSI model. This chapter describes the OSI model, and how it is an integral piece in the proposed solution presented in this thesis.

Chapter 5, Hardware interface addresses. This chapter explains specific technical information related to the hardware address and provides the foundation for understanding the proposed solution in this thesis.

Chapter 6, IEEE 802.11 wireless. This chapter provides a fundamental working knowledge of the IEEE 802.11 wireless protocol as well as several specifics required to understand the proposed solution in this thesis, such as operation modes and wireless radio headers.

Chapter 7, A solution to user tracking: Disposable hardware addresses. This chapter provides the solution to the problem posed in Chapter 1 and 2 regarding the use of the interface hardware address to track users.

Chapter 8, Testing. This chapter demonstrates the feasibility and ease of performing user tracking in the real world through implementation of a wireless radio sniffer on a single-board-computer.

Chapter 9, Testing results. This chapter analyzes the results of implementing and testing the hypothesis from the Chapter 8.

Chapter 10, Conclusion.

There are several topics that are not within the scope of this work and they include full descriptions of the OSI model, full descriptions of the IEEE 802.11 protocol, legal aspects related to privacy, RSSI triangulation, dataset cross-correlation and various computer and network security topics. The author may leverage previous work regarding these topics and will provide references to additional materials where the reader may go for further information.

CHAPTER 2

PREVIOUS AND RELATED WORK

Thesis-specific related work

There has been a significant amount of previous work done in the area of mobile privacy. In particular, [Gruteser] bears the most similarity to this work. In [Gruteser], the authors propose a solution to the problem where an adversary who controls several access points would be able to triangulate a client's position. In addition, similarly to this work, the authors note that interface identifiers uniquely identify each client, allowing tracking of location over time. They also proposes the same solution as this paper of enhancing user privacy through frequent disposal of a client's interface identifier. In some cases, the authors address the same problems as this author.

However, there are many differences between this work and [Gruteser]. In [Gruteser], the authors are motivated to mitigate the privacy threat of being able to uniquely identify and track a user's location throughout a network. Their proposed solution does not address passive information leaks as this work does even though their work does address the ability to track users based on active use of a wireless network. Additionally, [Gruteser] has a completely different, less-effective method of generating random addresses (thus preventing address collisions) than this work. Moreover, a large portion of the work in [Gruteser] was theoretical in nature and, therefore, a the "real-

world" application of their work was never realized. This work attempts to unite the theoretical analysis with "real-world" applications. In [Gruteser], the authors address integration with 802.1x, EAP-TLS and Radius, whereas this paper does not consider these protocols. Additionally, all of the described scenarios in [Gruteser] are based on active authentication, association and use of 802.11 wireless networks, while this work mainly considers passive probing of 802.11 networks.

A major enhancement of this work over the work of [Gruteser] is that the frameworks proposed by this research will also apply to all applications running on the system that could interrogate devices for their MAC address. To the application, this solution will be transparent and not affect normal execution, however if applications, advertising frameworks and tracking frameworks attempt to build off of or track interface addresses, they will cease working properly. This framework will leverage the application sandbox and application entitlements that are already available in Apple iOS, but this is a general framework that could be adapted to any mobile platform.

While this work draws many of the same conclusions as [Gruteser], it simultaneously identifies and addresses several additional problems. The first additional problem this work identifies is that it addresses the scenario of passive information leaks while using mobile devices. It is not necessary to be authenticated, associated or actively using a wireless network for tracking to be possible. In Chapters 7 and 8, it is demonstrated that passive information leaks can be collected from users without their knowledge and consent. The information collected can be processed and then used to correlate hardware addresses to behavioral actions in the context of location. The most common example of this is using the collected data to generate marketing strategies to be

used to market to individuals based on their interface address. The second addition this work makes is that it takes the theoretical ideas presented in [Gruteser] and analyzes them in a practical, "real-world" domain. Lastly, this work attempts to address some of the theoretical issues that were omitted in [Gruteser].

Another work that attempts to address some of the same issues as this work is [Li]. Li attempt to demonstrate that "the network flux over the sensor network provides fingerprint information about the mobile users within the field. Such information is exoteric in the physical space and easy to access through passive sniffing." They were able to identify mobile users within the network and instantly track their movements without breaking into the details of communicational packets.

In [Pang], the authors demonstrated that "users can be tracked using implicit identifiers, traffic characteristics that remain even when unique addresses and names are removed. Although we found that our technique's ability to identify users is not uniform —some users do not display any characteristics that distinguish themselves from others— most users can be accurately tracked. For example, the majority of users can be tracked with 90% accuracy when active often enough in public networks with 100 concurrent users or less. Some users can be tracked with even higher accuracy. Therefore, pseudonyms are insufficient to provide location privacy for many users in 802.11 networks."

There are many other excellent works on location, privacy and mobile devices including [Cooper, Tan, Lui, Yiu, Egele, Singelée, Lee, Kang, Zhang].

General mobile device privacy related work

In [Schlegel], the authors demonstrated the ability to deploy "a Trojan with few and innocuous permissions, that can extract a small amount of targeted private information from the audio sensor of the phone. Using targeted profiles for context-aware analysis, Soundcomber intelligently pulls out sensitive data such as credit card and PIN numbers from both tone and speech-based interaction with phone menu systems."

In [Felt], the authors show that even though Android has an advanced permissions system with a sandboxed execution environment, a "genuine application [can be] exploited at runtime or a malicious application can escalate granted permissions and imply that Android's security model cannot deal with a transitive permission usage attack and Android's sandbox model fails as a last resort against malware and sophisticated runtime attacks."

In [Seriot], the author illustrates that many of the attacks that occur on iOS have to do with theft or procurement of personal or private information. For example, iOS applications would have access to a user's address book that includes phone numbers, addresses and email addresses. Additionally, an application can also access other data such as call history, carrier information and photographs.

In [Bourimi], the authors presented a "privacy-respecting, indoor localization approach allowing better identification of shoppers' paths in stores." The system they created, Redpin, was designed to run on mobile devices. It consisted of two basic components: "1) a sniffer component (SC) that gathers and collects information about different wireless devices in range, in order to create a fingerprint, and (2) a locator

component (LC) that stores measured fingerprints in a shared repository and contains the algorithm to locate a mobile device." [Bourimi]

In [Gansemer], the authors present an algorithm for "RSSI fingerprint positioning based on Euclidean distance for the use in *a priori* existing larger and dynamically changing WLAN infrastructure environments." Similar to [Gansemer], in [Kitasuka], they "describe the design and implementation of a prototype of a proposed positioning system called WiPS. To improve the accuracy of relative location, WiPS uses RSSI between terminals."

Hardware address tracking concerns have been the subject of comments in other literature as well. For example, in [McCullagh], the author details concerns with a start up company called Euclid Elements that records the location of millions of smartphones. Euclid was launching a network that placed sensors in stores that passively detected Wi-Fi MAC addresses and stored them for later use. Currently, Euclid's model is an opt-out, not opt-in, model. Similar concerns to [McCullagh] are expressed in [Ribeiro]. Some companies, such as [Navizon], offer an indoor triangulation and positioning system using similar technologies as Euclid and Google.

CHAPTER 3

PRIVACY BACKGROUND INFORMATION

Broadly, user privacy violations deal with the exposing of confidential information. While privacy is the subject of many statutes, court opinions and regulations, for the purpose of this discussion, we will set aside the legal considerations (however, to demonstrate an application of the tort of invasion of privacy, *see*, *Lawlor v. North American Corporation of Illinois*, 2012 IL 112530 and the Restatement (Second) of Torts §652B). Confidential information can take many forms, *e.g.,* credit card numbers, names and passwords, birthdays, IP addresses, fingerprints and voiceprints and online purchase histories to list a few. All of this type of confidential information can be called PII, or Personally Identifiable Information. PII is information that is uniquely identified with a single person or can be integrated with other information to uniquely identify a single individual.

Types of Confidentiality Breaches

It is useful in impact analysis to categorize user privacy violations that occur by one (or more) of four categories. For example, it can help the analysis to assign a CWE (Common Weakness Enumeration) to a privacy violation. The [CWE] provides a

common language of discourse for discussing, finding and dealing with the causes of software security vulnerabilities as they are found in code, design, or system architecture.

Transparency

This category refers to the idea that a user should be notified when some process is accessing user data that might be considered sensitive. There should be an openness and honesty to the whole transaction. Ideally, this would be enforced at an operating system level so that all processes that wish to access user PII must go through a system API that would notify the user and notification could not be circumvented.

An example of this notification schema is present in Apple's iOS. When an application accesses the user's location data, the user is notified by placement of a small triangle icon in the status bar at the top of the device. This icon is present for as long as the process is active and alerts the user that the particular application is actively accessing the user's location data (see *Figure 1*). In addition, the user can determine, in "Settings" for iOS, which application(s) is/are currently accessing the user's location and which application(s) has/have accessed the user's location in the last 24 hours (see *Figure 2*). *Figure 3* shows what applications have accessed the users location in OS X.

Figure 1: Location services disclosure icon in Apple iOS operating system

Typically, transparency can be summarized by considering user expectations - "what would the user expect in this situation." When there is a lack of transparency, user reaction frequently includes surprise, fear and anger that the user's private information was accessed by a process without informing the user that the privacy invasion was occurring. Users are frequently and understandably worried that their information could be used for purposes other than the intended purpose of the application or that the information would not be properly safeguarded. Users also worry about additional personal information being accessed in the future without their knowledge if their past experience was that third party access was not disclosed.

Figure 2: Location privacy settings in Apple iOS operating system

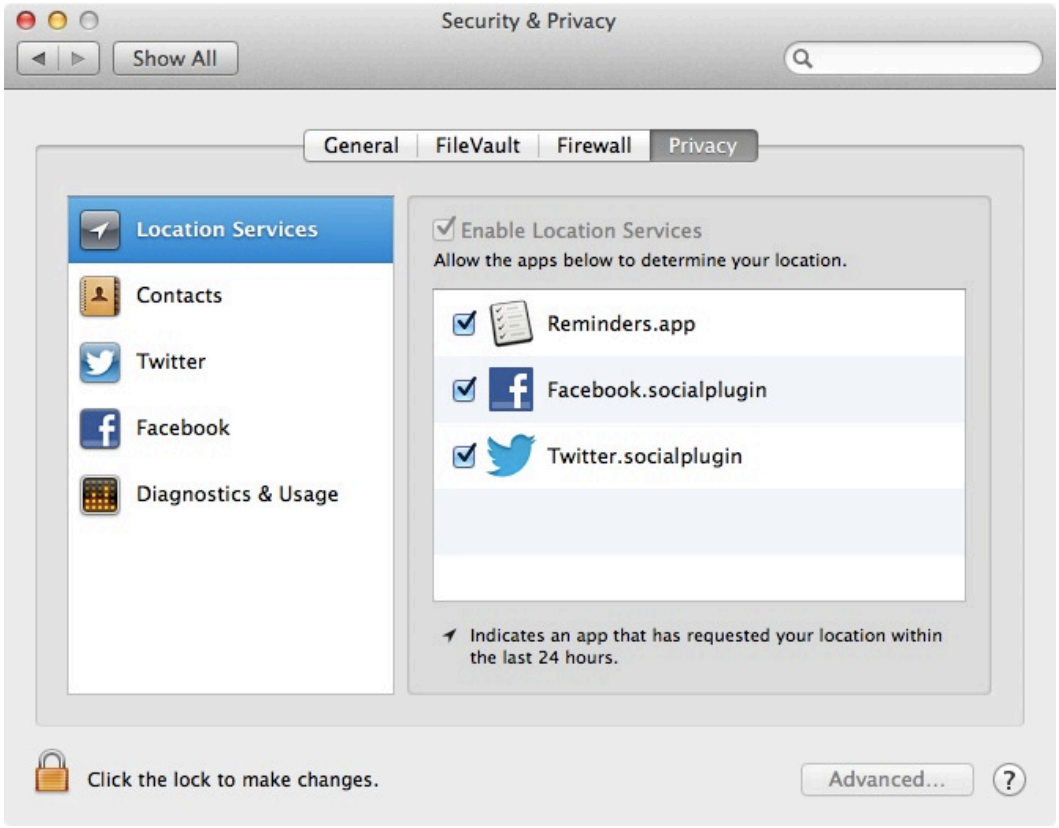

Figure 3: Location privacy settings in Apple OS X operating system

This "user distress" is demonstrated by the developmental history of the iOS application Path. In February 2012, the Path application was discovered to be uploading a user's entire address book. The upload occurred for the purposes of improving the quality of friend suggestions when the user used the "Add Friends" feature and to notify the user when one of the user's contacts joined Path. While the data was ultimately used for legitimate purposes, there was a lack of transparency regarding what was happening in the background of that process that caused controversy and a feeling of violation.

Consent

Transparency goes hand in hand with the second category of consent. This category refers to allowing the user to make an explicit choice of opting in or out of allowing a particular process to access personal information. When a user has given consent, the user is allowing access to that user's personal data. The option of consent is important and fundamentally fair because it gives the user a chance to give thoughtful consideration about allowing access. It allows the user to weigh the pros and cons of permitting access before allowing or denying access. The resulting "voluntary choice" action of granting or denying access is then clear and unmistakable and permits the user to weigh privacy concerns against potentially greater functionality. Again, ideally, consent requirements would be enforced at an operating system level – all processes that wished to access user data would have to go through a system API that would notify the user and could not be circumvented.

Once again, the Apple iOS provides an example of consensual choice. When an application attempts to accesses the user's location data, the user is notified with a modal dialogue. The user then is free to make the user's choice on whether to grant or deny the application access to this information (see *Figure 4*). *Figure 5* shows a consent dialogue on OS X.

One last important issue under the heading of consent should be the ability of the user to change the user's mind and revoke (or grant) consent at any time. For example, suppose the user originally granted access to a photo application to use the user's photo library. If the user initially trusted the third party but then learned that the application

was abusing private information, the user might decide to revoke photo library access to the application. A well- conceived operating system should provide for decisional modification on the part of the user. In iOS, the user can turn consent on or off in "Settings" (see *Figure 2*). In Android, users can change the settings to determine whether or not applications can access the user's current location (see *Figure 6*).

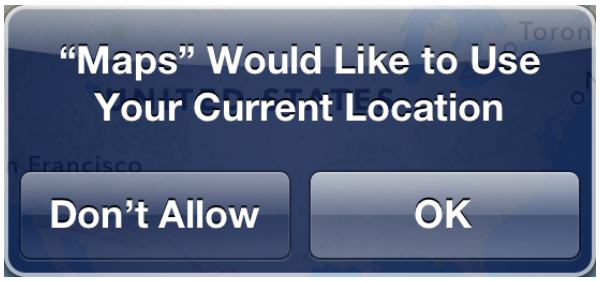

Figure 4: Location access consent dialog in Apple iOS operating system

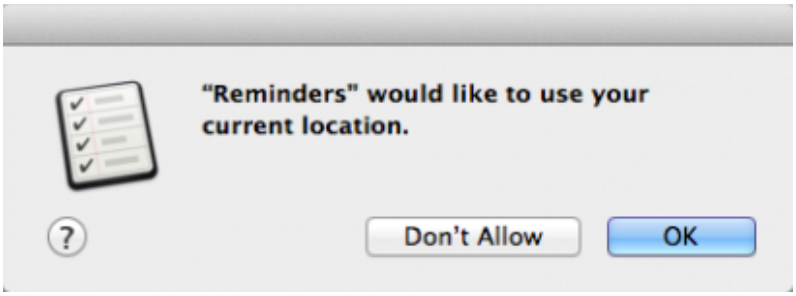

Figure 5: Location access consent dialog in Apple OS X operating system

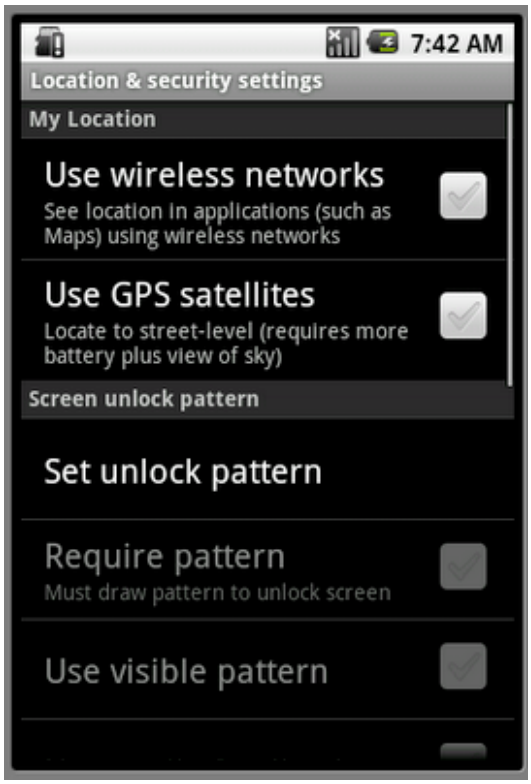

Figure 6: Location and security settings in the Android operating system

Over-access and over-collection

The third category of privacy violation is over-access/over-collection. This category of analysis dictates that the application should only access or collect data that is absolutely necessary to complete a task or provide functionality. While the user may have consented to allowing access to their private data and it is transparent this access is occurring, it is appropriate from a privacy perspective that the process only access information that is absolutely necessary to the task at hand.

For example, if the user grants a process access to the address book so that an application can implement a "Friend Finder" feature based on registered email addresses,

it would be poor privacy technique to also permit access to superfluous information (*e.g.,* phone numbers, addresses, *etc.*) that is also stored in the address book. The extra data collected would not be necessary to fulfill the functionality of the application. Another example would be the use and uploading of the exact coordinates of a user's location when a zip-code level accuracy would be sufficient to service the application. *Figure 7* shows the "Diagnostics and Usage" feature in OS X that allows Apple to receive crash and diagnostic information from users.

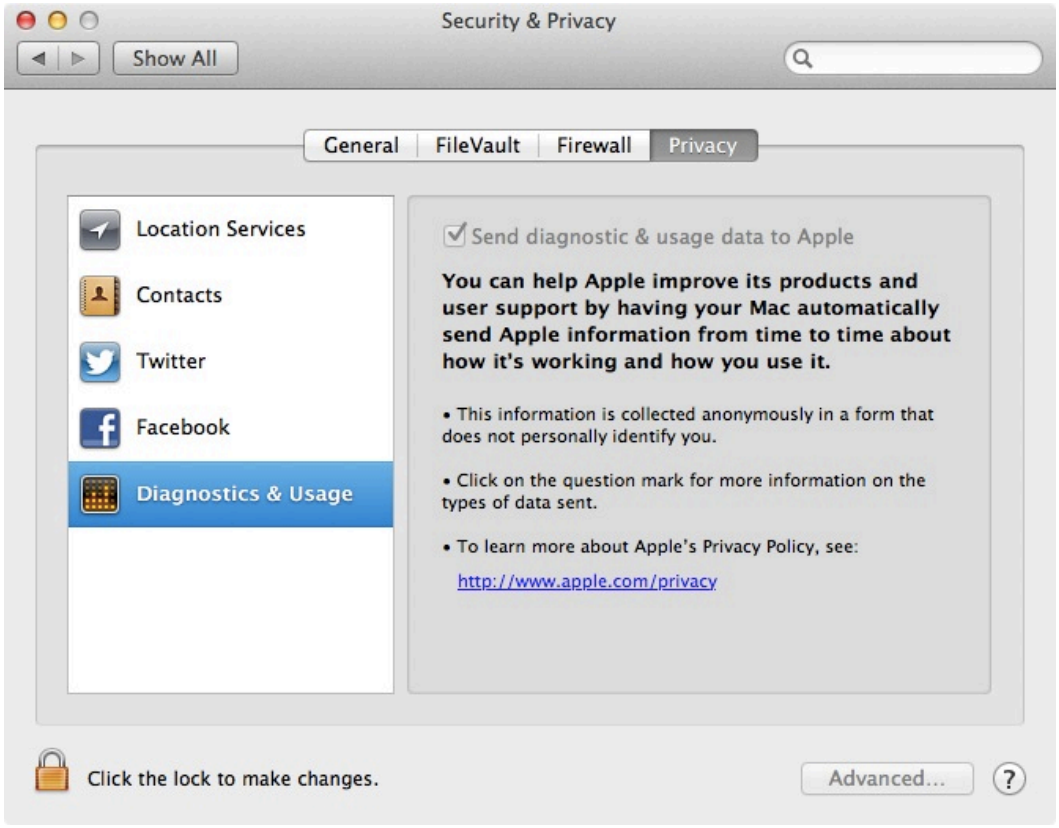

Figure 7: Diagnostic and usage settings in Apple OS X operating system

Yet another example would be a logging facility that collects error conditions from opted-in users. While the user has given consent to being included in the collection, personal user data does not need to be logged in order for the engineering department to fix application bugs. Consider the following three log samples:

In *Listing 1*, we see an error log message that is referring to the process `syncd` being unable to sync a photo on the network We are also able to learn some other information such as the exact date and time of the failure, the full path to the photo and the name of the wireless network the user is utilizing. A lot of this information is not necessary to fix or debug the underlying error condition.

```
10/7/12 11:55:57.814 AM com.foo.syncd: couldn't sync photo: “/Users/
jsmith/Photos/San Francisco Trip/IMG0234.jpg”, error: 4 no connection to
network “smith-ssid”, username jsmith.
```

Listing 1: Log file entry with poor user privacy

In *Listing 2*, we see an “improved” log message. In this example, the full file path was removed from the log message as well as the exact date and time being “fuzzed” to the month, day and hour. Also, the username has been removed from the log message.

```
10/7 11 AM com.foo.syncd: couldn't sync photo: “IMG0234.jpg”, error: 4 no
connection to network “smith-ssid”.
```

Listing 2: Log file entry with improved user privacy

In *Listing 3*, we see yet another “improved” log message over *Listing 2*. In this version, the log message has been “fuzzed” to a day of the week as well as completely

removing the filename and simply indicating the file type. Lastly, the log message does not include the network SSID.

```
Sunday com.foo.syncd: couldn't sync photo, filetype: JPEG, error: 4 no
connection to network.
```

Listing 3: Log file entry with best user privacy

In each of the last two examples of extracted information, the amount of information collected was reduced so that only the more essential information necessary to debug the problem remained. Additional ways to help reduce information collection to essential information include: 1) anonymization, 2) aggregation, 3) de-resolving, 4) minimization, and 5) decay of the information. Additional downsides to over-access/ over-collection include:

Additional cost for security measures to store data securely.

Additional cost for hard disk space to store the data.

Additional cost for staffing to administer these systems.

Additional cost of potentially needing to audit the data and developing retention plans for the data.

Additional cost of potentially responding to law enforcement or regulatory body requests for that data which, if not collected, cannot be subpoenaed.

Additional cost in having to parse out the relevant data from the irrelevant.

Data mishandling

The last category of privacy invasion is data mishandling. If a user has given consent to an application to access private information, while the application is transparent in what it is doing and accessing the minimum amount of data needed to complete its function, there is still the need for data security to ensure the safe transport and storage of the acquired data. All of the techniques above are wasted effort if private information is transmitted in the clear such that any attacker could intercept it.

A common method of securing the data is using encryption to secure the connection between two hosts. For example, when shopping online at Amazon.com, user data is encrypted with RSA encryption. While a full treatment of encryption is outside the scope of this work, there are some excellent resources relative to encryption such as [Anderson, Schneier, Bishop]. Encryption, and proper access controls, should also be used to secure the data at rest on the hard disk to prevent unauthorized parties from utilizing inappropriately accessed data or preventing access to the data in the first place.

However, encryption alone may not be enough to protect sensitive data. Consider the example of a corporate network that proxies all internal users' network connections. While the data is encrypted from the user's terminal to the proxy and from the proxy to the web host, the corporation could still invade the user's privacy as it holds the encryption keys for the proxy (this example pertaining to data to which the corporation does not have ownership rights).

An additional protection that could be used would be hashing PII. Consider again, the Path application example. Suppose that the application gained the user's consent to upload the user's address book. Before uploading the data, the application could hash the email addresses and phone numbers to further protect the sensitive information. This hashing technique would be improved by using some known salt. In cryptography, a salt value is a random number that is XORd with the key or password to harden the security measure from being cracked.

Severity Levels of Confidentiality Breaches

For classification reasons, the author has created several different levels of potential confidentiality breaches involving PII. This allows the author and other researchers to assign a logical classification documenting the seriousness of a particular breach. The seriousness of the vulnerability, in conjunction with the CWE classification, helps define whether or not the vulnerability is important enough fix and on what timeline. For example, a vulnerability that permits identity theft should be classified as a "high" level and should be fixed immediately. The confidentiality breach levels are defined as follows:

None: Having resulted in no exposure of confidential information.

Low: Having resulted in exposure of confidential information that could not directly be easily tied to the user, *e.g.,* device IDs that could be used to track users .

Medium: Having resulted in exposure of confidential information that is potentially harmful or would constitute harmful PII if integrated with other information.

High: Having resulted in exposure of extremely confidential information and harmfully reveals PII.

Potential Explanations for an Increase in Privacy Violations and Attacks

There are a number of reasons [Stites, Felt, F-Secure] that the community is experiencing an increase in privacy and confidentiality breaches,which include the following.

Increased computing power and storage capabilities

While many consumers may not recognize mobile devices as being equivalent in power to their larger counterparts (laptops and desktop PCs) due to their size, many smartphones, tablet devices and other PDA type devices have a rich set of hardware interfaces. Many smart phones, such as iPhones and Android-based phones, have powerful dual-core processors and a large amount of storage space to accommodate music, movies, documents and other types of media that can be consumed "on the go."

The available software applications that come pre-installed on the mobile device by the device manufacturer, such as web browsers, email clients and messaging applications, allow the user to much more readily interact with the physical and virtual

world than previous mobile devices. Furthermore, additional software applications can be downloaded, installed from the Internet and run by the user. These third-party applications are able to access the mobile device's advanced hardware as well as GPS and network interfaces (3G, WiFi and Bluetooth).

This avalanche of mobile devices and their increasingly capable software provide mobile malware and crimeware authors a much larger array of possibilities to carry out their attacks. In addition, more sophisticated hardware and software possibly make it easier for these attackers to "hide" their attack by ensuring that it only consumes a small portion of the resources, thereby modeling a legitimate application.

Increased network connectivity

There is a widespread availability of 802.11 WLANs and high-speed broadband data access (3G, WiMAX) compared to only a few short years ago. These services allow users to stay constantly connected to services such as email and messaging at home, at work and in public places such as coffee shops. Many applications utilize network connections to either request or send data and present additional vulnerability. For example, a game application might transmit a user's high score to a web server for storage. Additionally, users are demanding that previously inaccessible data be opened up to remote access and this forces providers of services to provide such access or lose out to a competitor who does provide such access.

There are many examples of the vulnerabilities introduced by increased network connectivity. Many applications, such as the Amazon.com shopping application, rely on

the fact that the cell phone will have a network connection to receive and send data. A more recent development is that many applications utilize location-based services, such as Facebook, Twitter, and FourSquare. These applications provide additional functionality if they are able to access the network and a user's current location but do so at the risk of putting more information where it could potentially fall into the wrong hands. Lastly, many applications can make use of social data, such as the friends one might have on Facebook. This Facebook data could be stored within the application. While Facebook might maintain rigorous security standards on who and what can access the user's data, other third party applications might not be so careful with the user's data.

Standardization of operating systems and interfaces

The OS can be made consistent on any device in the same family of devices, so malware applications would have more effect, being able to exploit the same security vulnerability across many devices. Additional vulnerability may, surprisingly, be unintentionally assisted by the device manufacturers themselves. Many manufacturers give third party developers access to the system to write applications for the platform. For example, one can freely download the Android and iOS SDK. Using this provided SDK, those writing malware can craft a virus or some other piece of malware and then submit it for inclusion in the appropriate application storefront. The "implied safety" of the application being available in the storefront can provide users with a false sense of security that the application is malware-free.

Enterprise integration

Many mobile devices, such as Android, iOS and Blackberry, support standards that permit integration into an enterprise environment. For example, many of these devices have support for Virtual Private Networks (VPNs) as well as Exchange server integration. Thieves and malware authors recognize that this integration will greatly enhance the infection potential if a mobile virus, Trojan horse or worm is able to spread from a mobile device to a corporate environment.

Consider the case of an employee with a mobile device becoming infected while at a coffee shop. The employee takes his infected device back to the corporate environment where the it could spread that infection throughout the organization. Where previously an attack might have only stolen information pertaining to the particular victim's mobile device, now the attacker could potentially obtain information associated with many different people as well as corporate information.

Other reasons, social engineering and hacktivism

The number of socially engineered attacks are becoming more prevalent and more sophisticated. Use of malware to express displeasure or promote a group's or individual's personal beliefs ("hacktivism") represents an increasingly attractive method to some to register protest. Hacktivism has become part of the mainstream in 2011 due to groups such as Anonymous and LulzSec."

Privacy by Design

Since users are storing more personal information in their devices and the number of privacy-related violations are increasing, a good way to mitigate potential privacy issues is to design systems with privacy in mind. Too often, privacy is an after-thought of developers. "Privacy by design" means that privacy and data protection are "embedded through the entire life cycle of technologies, from the early design stage to their deployment, use and ultimate disposal [European Commission]." This process can be directly embedded into a software development lifecycle methodology. To support this initiative, the United State Federal Trade Commission released a report detailing five action items that a privacy by design framework should support:

Do Not Track: "The browser vendors have developed tools that consumers can use to signal that they do not want to be tracked [FTC]."

Mobile: "The Commission calls on companies providing mobile services to work toward improved privacy protections, including the development of short, meaningful disclosures [FTC]."

Data Brokers: "To address the invisibility of, and consumers' lack of control over, data brokers' collection and use of consumer information, the Commission supports targeted legislation that would provide consumers with access to information about them held by a data broker [FTC]."

Large Platform Providers: Exploring privacy and other issues related to comprehensive tracking by large platforms, "such as Internet Service Providers, operating systems, browsers, and social media seeking to comprehensively track consumers' online activities [FTC]."

Enforcing Self-Regulatory Codes: "The Department of Commerce, with the support of key industry stakeholders, wants to undertake a project to facilitate sector-specific codes of conduct and continue to enforce the FTC Act to take action against companies that engage in unfair or deceptive practices [FTC]."

CHAPTER 4

THE OSI MODEL

The OSI model is a standardized architecture for communications systems described in abstract layers. In this model, similar communication functions are grouped into logical layers. Each layer in the model works with the layer above it and below it to incrementally prepare the data to be in a standardized format that permits transmission (when sending data) or utilization (when receiving data) once all layers are traversed. When an application sends data, the data progressively works down the stack from the application layer to the physical layer and when data is received, it makes its way up the stacked layers to the application. Data is touched by each layer – no layer is skipped. Additionally, the ordering of these layers operating on the data is strict. There are 7 layers in the model (see *Figure 8*).

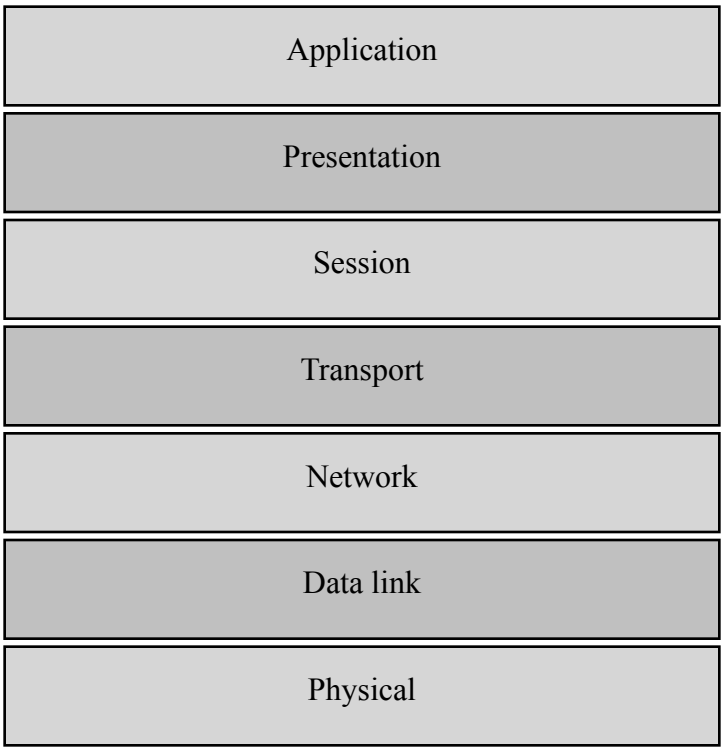

Figure 8: ISO OSI network model

The privacy solution the author is proposing is concerned with the physical and data link layer, however there would need to be coordinated support in higher layers, such as in the application software (the application layer). While a full discussion of the OSI model is outside the scope of this paper, the reader can easily find additional authoritative resources describing and discussing the structure and implications of the OSI model and networks in [Zimmerman, Day, Stallings, Leon-Garcia]. Because the privacy solution proposed in this paper frequently depends upon implementation at the most basic levels of the OSI model in order to improve the possibility that privacy issues will not be ignored, a review of the capabilities of each layer is in order.

OSI Layer 1: Physical

Layer 1, the physical layer, is the lowest layer in the OSI model and represents the carrier of the signals. This layer provides several key pieces of functionality:

Establishing physical transmission mediums

This is the most basic of the physical layer requirements. There must be some form of transmission medium for the analog signal to propagate through from source to destination. In IEEE 802.3 Ethernet, the physical layer would be twisted pair copper wiring [IEEE Ethernet]. In IEEE 802.11a/b/g/n, the physical layer would be the surrounding air [IEEE Wireless]. In both of these example, the transmitted data would traverse this layer from the sender to reach its destination.

Topology

Network topology refers to the arrangement of hosts on a network. This arrangement can be physical or logical. In the case of physical topology, this refers to the actual "real-world" placement of the network host. In the case of logical topology, this refers to how data flows within a network, regardless of the network design or host placement. While the physical distance between nodes can be feet, yards, miles or hundreds of miles, this does not change the flow of data within the network. Other factors such as the operating system, physical connections and other physical considerations may differ, yet the topology of a network would remain the same [Leon-

Garcia, Stallings]. Examples of network topologies include bus, point-to-point, star, ring and mesh [Leon-Garcia, Stallings].

Hardware specifications

This aspect of the physical layer also specifies the technical functionality of network connectors, terminators, network cables, radios and interface cards. This layer also includes all the interconnection hardware used in the network, such as routers, hubs, switches, *etc*.

Encoding and decoding data

This component of the physical layer handles the conversion of data from digital form to the corresponding analog signal that is transmitted over the communication channel by the source host and the conversion back to digital form on the receiving host end.

OSI Layer 2: Data link

Layer 2, the data link layer, is served by the physical layer and serves the network layer. This layer provides several pieces of functionality:

Addressing

The data link layer contains the functionality of addressing. Each device on a network has a single, unique identifier called the hardware address or MAC address. This address is used by the data link protocols to ensure that data is received by the intended machine properly. The hardware address is also plays a key role in this work and will be discussed in more depth in the subsequent chapters.

Media access control

Media access control functionality attempts to manage access to the shared transmission medium. There are different algorithms and rules that govern this access [Leon-Garcia, Stallings]. For example, on Ethernet, all hosts are connected through wiring. If two hosts attempt to use the transmission at the same time, there will be a conflict in access and data will be corrupted. [IEEE Ethernet] uses CSMA/CD (Carrier sense, multiple access/collision detection) to avoid and recover from conflicts.

Error handling

The data link layer will attempt to confirm data integrity, typically through the use of a cyclic redundancy check (CRC). Depending on the recovery algorithm, it is possible to recover from transmission errors with this functionality.

Frame synchronization

Frame synchronization functionality arranges bits from the physical layer into frames. A frame is a data packet. Typically, each frame begins with a preamble or start sequence. A frame also contains a data payload and ends with a CRC that can be used to check the integrity of the data transmitted. The frame allows for the layers to create a logical unit of transmission.

Some examples of layer 2 protocols would be 802.3 Ethernet, 802.11a/b/g/n MAC/LLC, DOCSIS, PPP and Token Ring [Leon-Garcia, Stallings].

OSI Layer 3: Network

The third layer, the network layer, is served by the data link layer and serves the transport layer. This layer is mainly concerned with two functions: 1) defining how inter-connected networks function and 2) providing the functionality of transferring data between network nodes. To support these goals, the network layer has some specific responsibilities:

Provision of logical addressing services

Each host on a network needs a unique logical address to which data can be delivered. While the link layer has addressing capabilities, those addresses refer to physical devices, whereas the logical addresses in the network layer are hardware

independent and refer to nodes of the network. Internet Protocol (IP) is one such protocol for addressing nodes.

Routing services

Layer 3 also provides the ability to move data across inter-connected networks and routing the data to the proper destination. To achieve this goal, the overall data transmission is broken into discrete packets, each of which has a network layer header that functions as an address, and the entire data transmission is then reassembled on the receiving end when all packets sent are received at the address specified in the network layer header.

Fragmentation and reassembly

Different link layer protocols have different link layer frame sizes. For example, a regular Ethernet frame has a maximum payload of 1,500 bytes [IEEE Ethernet]. Since this layer provides service to the link layer, it must break down sequences of data into chunks that will fit inside Ethernet link layer frames. This process of breaking down data for transmission is called fragmentation. Conversely, when the data arrives at the destination host, the link layer passes data to the network layer, which then has the responsibility for reassembly.

Error handling

There are protocols that allow hosts to determine the amenability of the recipient to being contacted or the status of other hosts on the network as well as the ability to handle errors in transmission.

OSI Layer 4: Transport

The fourth layer, the transport layer, is served by the network layer and serves the session layer. The transport layer "provides end-to-end communication services for applications [Braden]." The transport layer provides arguably some of the most important functionality in the OSI model, allowing multiple processes on the same host to access the network in an efficient, reliable manner.

Connection-oriented communication

Up until this layer, all of the models were connection-less, meaning they sent data in the form of packets from one end point to another. This layer introduces the option of having connection-oriented communication, meaning that the end points establish an actual connection before transmitting any user data. This connection-oriented model allows for the subsequent properties of this layer. An example transport protocol is TCP (transmission control protocol) and UDP (user datagram protocol).

Reliability

Since lower layers are connection-less, the delivery of packets was by best effort up to this point. For example, it is possible that data may be corrupted during transmission or lost during periods of high network congestion. This layer offers the ability to add reliability or error correction to a connection-oriented protocol by allowing endpoints to acknowledge receipt of data packets or request re-transmission of packets that were not received or discarded due to bit errors.

Flow and congestion control

Due to the fact that a network may be comprised of many different types of hosts, there is a need to adapt the connection protocol to the host. For example, some hosts may be machines that don't have a lot of processing, memory or network bandwidth resources. If one endpoint were to have an extremely high data transmission rate, then it might be possible to overwhelm a slower receiving endpoint causing the data to overflow its receiving buffer. This layer introduces the ability to add flow control such that a receiving host can control the rate of transmission by the sending host. An example of a congestion control algorithm is "slow-start" and one example is detailed in [Hung].

In-order delivery

Network conditions can be unpredictable; for example, there can be a high amount of congestion in some network segments and packets could take different paths through a network to reach their destination. Due to the fact that none of the previous

layers guarantee in-order delivery (and it is generally a requirement for applications to have in-order delivery), another function of the transport layer is to re-order the packets for proper delivery. This is typically done through sequence numbers [Leon-Garcia, Stallings].

Multiplexing

The last major responsibility of the transport layer is to ensure that all processes on a host have access to the network. Since computers are able to handle many different processes at once, it is necessary to have an abstraction that will support each process having access to the network at the same time. This is commonly achieved through an abstraction such as defining a port or socket through which an application should send and receive its data [Leon-Garcia, Stallings].

OSI Layer 5: Session

The fifth layer in the OSI model is the session layer. This layer is served by the transport layer, serves the presentation layer and "provides the control structure for communication between applications; establishes, manages, and terminates connections (sessions) between cooperating applications [Stallings]." This layer is necessary to "organize and synchronize dialogue and to manage the exchange of data [Stallings]." Some additional functionality that can be found in this layer supports authentication, authorization and error recovery. In regard to error recovery, "[t]he session layer can

provide a checkpointing mechanism, so that if a failure of some sort occurs between checkpoints, the session entity can retransmit all data since the last checkpoint [Stallings]."

OSI Layer 6: Presentation

The sixth layer in the OSI model is the presentation layer. This layer is served by the session layer and serves the application layer. The presentation layer is mainly responsible for "providing independence to the application processes from differences in data representation [Stallings]." This permits for the network to accommodate a number of different hosts all of whom can functionally utilize the network. For example, there can be any number of different hosts on a network ranging from Windows PCs, Macintoshes, and UNIX and Linux hosts. These systems all handle and represent data in different ways and the presentation layer hides this complication and makes it appear as the network is homogenous.

This layer "defines the format of the data to be exchanged between applications and offers application programs a set of data transformation services. The presentation layer also defines the syntax used between application entities and provides for the selection and subsequent modification of the representation used. Examples of specific services that may be performed at this layer include data compression and encryption [Stallings]."

OSI Layer 7: Application

The last, highest layer in the OSI model is the layer that is closest to the user - the application layer. "The application layer provides a means for application programs to access the OSI environment. This layer contains management functions and generally useful mechanisms that support distributed applications. In addition, general-purpose applications such as file transfer, electronic mail, and terminal access to remote computers are considered to reside at this layer [Stallings]." It is worth noting however, that not all uses of the application layer come from applications; it is possible for the OS to use the application layer as well.

CHAPTER 5

HARDWARE INTERFACE ADDRESSES

The hardware address is a globally unique identifier that has the purpose of uniquely identifying a network interface on a physical network segment such that inter-device communications are routed to the correct device. Hardware addresses are used in many networking standards, such as IEEE 802.3 (Ethernet), IEEE 802.11 wireless networks, IEEE 802.5 (token ring), ATM, FDDI and Bluetooth. When one is referencing the OSI model, a hardware address resides in layer 2.

A network interface's hardware address is typically burned into the hardware or is stored in read-only firmware on the device. In the IEEE MAC-48 standard [IEEE Wireless], the identifier is 6 bytes long (48-bits). The address is organized into octets, where the first 3 bytes of the address are the OUI or the Organizationally Unique Identifier. The OUI indicates what manufacturer "owns" that particular address space. A full OUI list is available at [IEEE OUI].

The last 3 bytes of the MAC-48 identifier are the network interface ID, that is effectively a serial number or unique identifier within the OUI address space. The last 3 bytes may be assigned by the manufacturer in any way it desires, with the only constraint being uniqueness. In theoretical total, this MAC-48 address space supports 2^{48} (281,474,976,710,656) addresses. An example of a hardware address could be

`FF:FF:FF:FF:FF:FF`. In this particular example, this hardware address is the broadcast address, since all the bits are 1.

Hardware addresses can be divided into two categories: universally administered and locally administered. This description indicates if the address is considered globally unique and unchanged or if the settings were overridden locally and changed to a different address, respectively. To indicate the category in which the address falls, the "U/L bit" is set to 1 if the address is locally administered. By standard, the U/L bit is the second-least significant bit of the most significant byte of the address (see *Figure 11*).

During normal operation, network traffic is only delivered to a receiving device if the hardware address in the data packet matches the hardware address of the receiving device. Non-matching traffic is discarded unless the receiving device is in promiscuous mode, described below.

In TCP/IP networks, the hardware address of an interface can be queried if one knows the IP address of a particular node. ARP, Address Resolution Protocol, translates the IP address into a hardware address that uniquely identifies each node in a segment, allowing frames to be delivered to the correct receiving device. In iOS, a user can see their MAC address in Settings under General / About (see *Figure 9*). In OS X, a user can see their MAC address in Preferences under Network / Advanced / Hardware (see *Figure 10*) or by using the `ifconfig` command. In Windows, the user may find this information by using the `ipconfig \all` command in DOS.

Figure 9: iOS Network Preferences User Interface

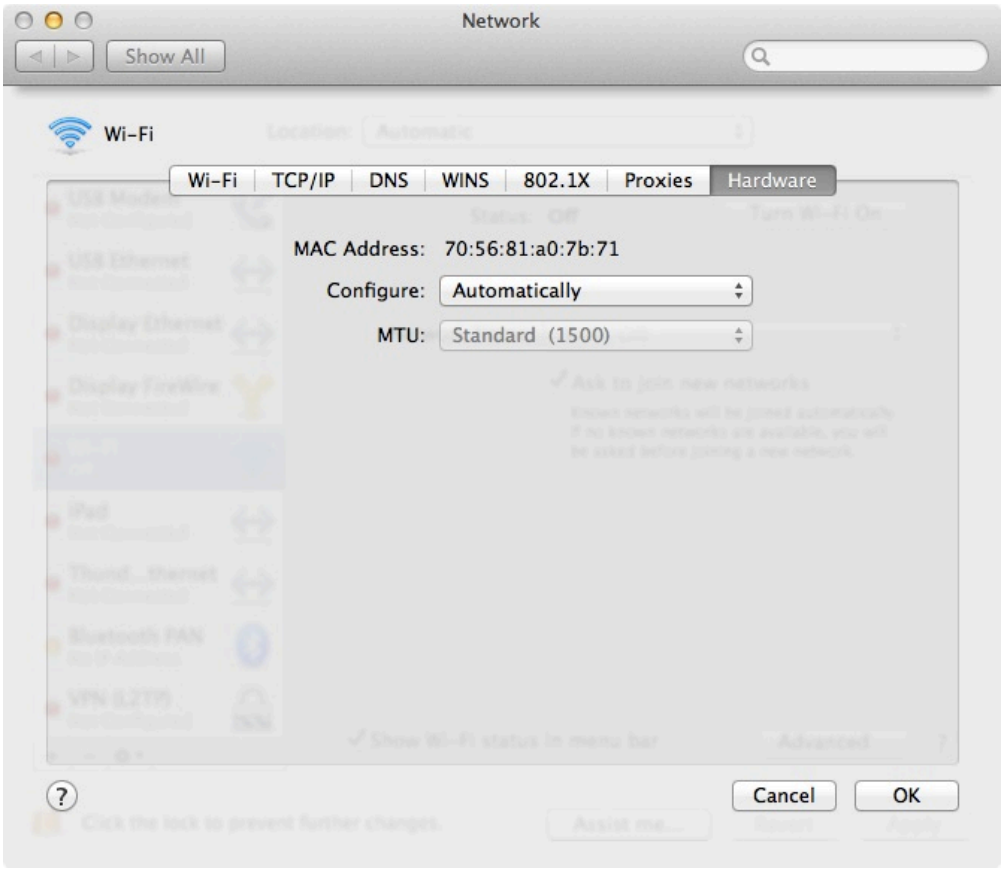

Figure 10: OS X Network Preferences User Interface

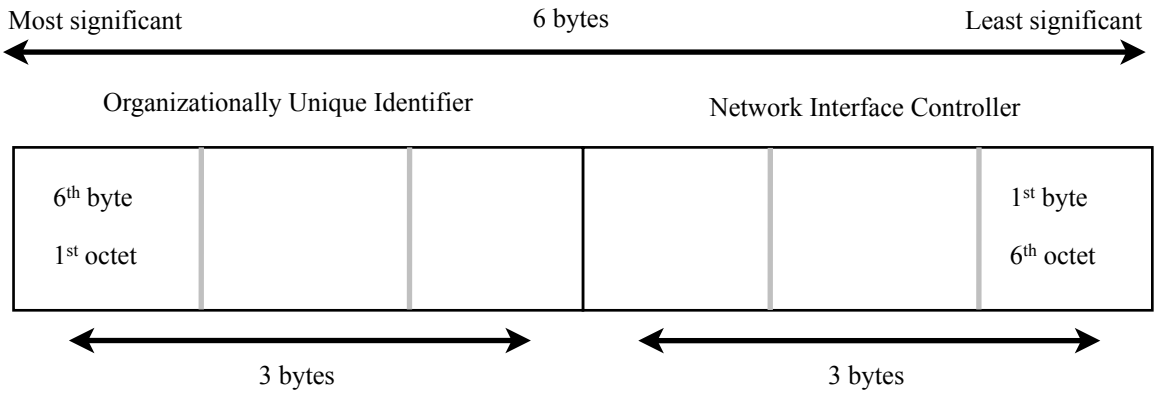

Figure 11: Hardware address diagram

CHAPTER 6

IEEE 802.11 WIRELESS

802.11 Background Information

IEEE 802.11 is the set of standards and specifications that governs and controls development of wireless local area networks and wireless communication in the 2.4, 3.6 and 5 GHz frequency bands [IEEE Wireless]. These standards are controlled by the IEEE LAN/MAN Standards Committee. The original 802.11 specification was published in 1997 [IEEE Wireless], however this standard is now obsolete. The current standard as of this publication is 802.11-2012 and there are many amendments that have to do with additional, add-on features. While a full, in-depth discussion of this standard is far outside the scope of this work, one can refer to [IEEE Wireless, Leon-Garcia, Stallings] for more information. This work will briefly include some basic background information related to IEEE 802.11 communication that is pertinent to this paper.

Basic 802.11 Operation

There are a number of steps that a station will go through to join an 802.11 wireless network. While a full discussion of all topics is outside the scope of this work, one may find more information at [IEEE Wireless].

Scanning

Scanning is the most basic operation that a station can perform when preparing to join a wireless network. Scanning is typically triggered by the user in some sort of user interface, but does not have to be user-initiated. There are two types of scanning: active and passive. In active scanning, a station is transmitting Probe Request frames and waiting for Probe Response frames. These Probe Request frames can be broadcast or directed at particular BSSIDs and are meant to query the access points. The Probe Response frame is returned with the SSID of the of the wireless network in addition to other technical information. Based on the information in the Probe Response, the station can then decide which networks are suitable for the station to join. In passive scanning, stations simply listen for transmitted Beacon frames that contain the name of the SSID and additional technical information about the network. The station can then decide which networks are suitable for the station to join. If encryption is supported by the access point, it is not in effect at this point.

Synchronization

An additional function of Beacon frames is that it allows stations to synchronize with the access point to support additional functionality. For example, to support a power-saving mode, where the wireless radio can sleep for a time period and use less battery power, wireless clients have to synchronize their clocks with the clock for the access point. To do this, the stations use a timestamp that is transmitted with the Beacon frame and then, accordingly, adjust their clocks by some calculated offset. One last function of Beacon frames is that they can contain capability information, such as supported transmission rates, channels and encryption types available. If encryption is supported by the access point, it is not in effect at this point.

Authentication

Before the station can associate to the network, the access point must authenticate the client. This step serves to establish the identity of a client. There are several basic types of authentication: 1) open system and 2) shared key authentication. In an open system, there is no authentication mechanism, meaning anyone can join this access point. The client sends an authentication request to the access point, the access point authenticates the client and sends back an authentication response after which the client joins the network. This type of network has no encryption and is insecure.

In a shared key environment, the client will send an authentication request to the access point which will respond with a “challenge” in order to verify identity. The user

typically will enter a password and the client will encrypt this information and send it to the access point which will validate the appropriate response to the challenge. If the challenge is validated, the access point will authenticate the client and send back a response. If the challenge is not validated, the access point will deny network access until the challenge is properly validated (the access point may support limited responses or unlimited responses, as it is programmed). If encryption is supported by the access point, it is not in effect at this point.

Some examples of a shared key authentication system include WEP (Wired Equivalent Privacy), WPA (WiFi Protected Access) and WPA2 (WiFi Protected Access 2). WEP is currently not recommended to secure a WLAN because of a weakness that will allow a cracker to capture an encrypted form of an authentication response frame and use cracking software to derive the WEP key [Bittau].

Association

The last step in a client joining a wireless network is the Association Request and Response. This step will enable a client to send and receive data on the network. The client sends an Association Request to the access point and it will either allow or deny the client in an Association Response. If the station's access is allowed, the access point will add the client in its list of connected clients and will begin to forward packets to the client. If encryption is supported by the access point, it is in effect at this point.

This process can be accurately described using the IEEE 802.11 state machine [IEEE Wireless] (see *Figure 12*). At any point, the station can progress through the states

or be stuck in a particular state. A station may leave a network by sending a Disassociation or De-authentication Request.

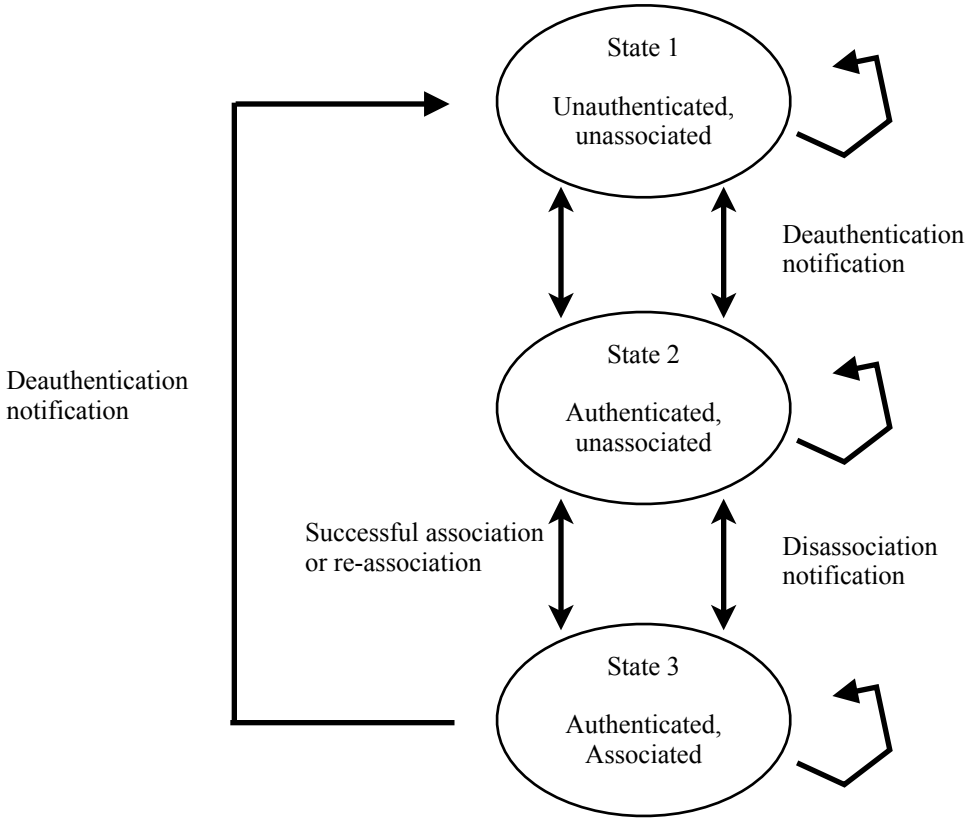

Figure 12: IEEE 802.11 state machine

Once a station has fully associated with a wireless network, it is free to receive and transmit data using CTS & RTS management frames.

Wireless Security and Potential Risks of Wireless Networks

While a full discussion of wireless security is outside the scope of this paper it is worth mentioning that the two biggest risks to privacy in wireless networks include network intrusion and wireless cracking and data interception. In both cases, the risks can be mitigated by using some sort of encryption or wireless security protocols such as

WEP, WPA, WPA2, VPN or 802.1X. It is worth noting that the use of WEP is considered insecure due to the ease of cracking the encryption key. Additionally, many consumer brand routers such as Cisco, DLink and NetGear, offer "security" features such as hidden SSIDs and MAC address authentication. These features should not be considered security mechanisms due to the relative ease with which they are bypassed. More information wireless network vulnerabilities and security mechanisms can be found in [Leon-Garcia, Stallings].

IEEE 802.11 Operation Modes

Infrastructure mode

This is the typical operation mode of 802.11 stations, STAs, and access points. In Infrastructure mode, clients assume Managed mode while the access points (APs) assume Master mode. In Managed mode, all the clients are connected to a main access point. The AP will manage these clients. As a collection of client nodes and access point, this unit is called the Base Service Set or BSS. Each BSS is identified by a BSSID, which corresponds to the access point's MAC address.

Multiple BSSs can be linked together to form an Extended Base Service Set or EBSS. An EBSS is typically created when a single access point cannot cover an entire geographic location due to remoteness and lack of signal strength, such as in a large office building. This allows the user to roam freely while the user's antenna switches transparently between the stations based on the signal it receives.

Ad hoc mode

In an ad hoc mode, stations (STA) connect to each other to form a peer-to-peer network. There is no central access point in ad hoc mode, but rather each station acts as both an access point and a client.

Promiscuous mode and monitor mode

In the 802.11 standard, the physical layer is shared between STAs. Therefore, if a client broadcasts in a particular area, it is sharing that medium with all other STAs that are within its range. In normal operation, the wireless radio driver will filter out any packets that are not addressed to the STA, as indicated by the hardware address, and drop them. In Promiscuous mode, all of the packets are delivered to the client, regardless of to whom they were addressed. This allows a client to capture all the traffic on a network segment. While this functionality can be used to audit and inspect networks for proper operation, it can also be used for traffic sniffing.

Monitor mode is similar to promiscuous mode but adds the ability that clients can capture packets without having to be associated with an access point or ad hoc network first. Monitor mode only applies to wireless networks, while promiscuous mode can be used on wired or wireless networks. Again, like Promiscuous mode, Monitor mode can used for both legitimate purposes, such as network audits, and also for malicious purposes, such as packet injection, traffic sniffing and WEP cracking. Typically the

support for both Promiscuous and Monitor mode are dependent on the hardware and driver support.

One important note to make about promiscuous and monitor modes is that these modes are unaffected by either the unicast or multicast bit (the least significant bit of the most significant address octet) setting. The traffic will be delivered to the intercepting station regardless of this setting. *Listing 4* shows how someone can easily capture a MAC address and place the interface in promiscuous mode.

```
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/if.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>

int main(int argc char *argv[]) {
  struct ifreq s;
  int fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_IP);

  #ifdef __APPLE__
    strcpy(s.ifr_name, "en0");
  #else
    strcpy(s.ifr_name, "eth0");
  #endif
  if (0 == ioctl(fd, SIOCGIFHWADDR, &s)) {
    for (int i = 0; i < 6; ++i) {
      printf("%02x", (unsigned char) s.ifr_addr.sa_data[i]);
    }
    puts("\n");
    return 0;
  }

  ioctl(fd, SIOCGIFFLAGS, s);
  s.ifr_flags |= IFF_PROMISC;
  ioctl(fd, SIOCGIFFLAGS, s);

  return 0;
}
```

Listing 4: C code to modify a network interface's promiscuous mode setting

Wireless Radio Headers

Wireless radio headers serve two major purposes. The first purpose is that they are a preamble to the rest of the wireless frame. Frame preambles are used to synchronize communications between two or more systems. When a preamble is seen, then stations know that a block of data is about to be transmitted. This also ensures that proper timing is observed so that all systems are interpreting the start of data transmissions correctly. Preambles add to the overhead of transmitting a packet.

Overhead is metadata that supports the transmission of user data and does not contribute to the actual user message. However, the cost of the overhead of a transmission can be amortized by the size of the packet that is transmitted. If the payload of the packet is small, then the header contributes a higher percentage of overhead. If the payload of the packet is big, then the header "cost" is reduced because the percentage of overhead is much smaller. A high percentage of overhead is bad for a network because instead of transmitting useful data, much of the bandwidth is consumed with "overhead" data. This leads to a decrease in throughput and bandwidth. One may calculate overhead costs

$$Overhead = \frac{FrameSize - PayloadSize}{FrameSize} \tag{1}$$

and protocol efficiency.

$$Efficiency = \frac{PayloadSize}{FrameSize} \tag{2}$$

The second use of wireless headers is that they are a mechanism to supply information from the driver about wireless frames to user-space applications and from user-space applications to the driver. This information can be useful in reporting characteristics about the frames, such as the channel, data rate, and the RF signal power and noise level at the antenna. There are several common 802.11 header types:

Radiotap

"Radiotap is the defacto standard for 802.11 frame injection and reception [Radiotap]." The system was initially designed for NetBSD systems by David Young. The radiotap format is more flexible than the Prism or AVS header formats because radiotap format allows driver developers to specify an arbitrary number of fields based on a bitmask presence field in the radiotap header. This flexibility allows new fields to be added over time without the need to change existing parsers to support them.

Each radiotap capture starts with a radiotap header (see *Listing 5*).

```
struct ieee80211_radiotap_header {
       uint8_t it_version;
       uint8_t it_pad;
       uint16_t it_len;
       uint32_t it_present;
}; __attribute__((__packed__));
```

Listing 5: Radiotap header struct

This header contains information such as the version of the header (as of publication, the version is 0, but can change for drastic revisions to the header), the length of the header, including data fields, in bytes, and what fields are currently present in the

header. The `it_present` variable is a bitmask of the radiotap data fields that follow the header. By setting bit 31 using the mask `0x80000000`, driver authors can extend the presence bitmap by another 32 bits. This can be recursively done to chain long segments of header data together. To reduce overhead cost, if a field is not present in the bitmask, then the field simply does not exist in the header data and the data is not padded out. To keep natural byte alignment, padding is added to some fields so that data is aligned on natural word boundaries. As of publication, the currently available fields are shown in *Listing 6*.

```
enum ieee80211_radiotap_type {
       IEEE80211_RADIOTAP_TSFT = 0,
       IEEE80211_RADIOTAP_FLAGS = 1,
       IEEE80211_RADIOTAP_RATE = 2,
       IEEE80211_RADIOTAP_CHANNEL = 3,
       IEEE80211_RADIOTAP_FHSS = 4,
       IEEE80211_RADIOTAP_DBM_ANTSIGNAL = 5,
       IEEE80211_RADIOTAP_DBM_ANTNOISE = 6,
       IEEE80211_RADIOTAP_LOCK_QUALITY = 7,
       IEEE80211_RADIOTAP_TX_ATTENUATION = 8,
       IEEE80211_RADIOTAP_DB_TX_ATTENUATION = 9,
       IEEE80211_RADIOTAP_DBM_TX_POWER = 10,
       IEEE80211_RADIOTAP_ANTENNA = 11,
       IEEE80211_RADIOTAP_DB_ANTSIGNAL = 12,
       IEEE80211_RADIOTAP_DB_ANTNOISE = 13,
       IEEE80211_RADIOTAP_RX_FLAGS = 14,
       IEEE80211_RADIOTAP_TX_FLAGS = 15,
       IEEE80211_RADIOTAP_RTS_RETRIES = 16,
       IEEE80211_RADIOTAP_DATA_RETRIES = 17,

       IEEE80211_RADIOTAP_MCS = 19,
       IEEE80211_RADIOTAP_AMPDU_STATUS = 20,

       IEEE80211_RADIOTAP_RADIOTAP_NAMESPACE = 29,
       IEEE80211_RADIOTAP_VENDOR_NAMESPACE = 30,
       IEEE80211_RADIOTAP_EXT = 31
};
```

Listing 6: Radiotap data field bitmask

Typically, support for radiotap is found on BSD operating system variants, however there is support for Linux depending on the kernel version and drivers available. Other options for wireless headers include AVS and Prism headers, for which similar information that is available in radiotap, is available. For more information on the radiotap header format, please refer to [Radiotap]. *Figure 13* shows an example packet capture with a radiotap header.

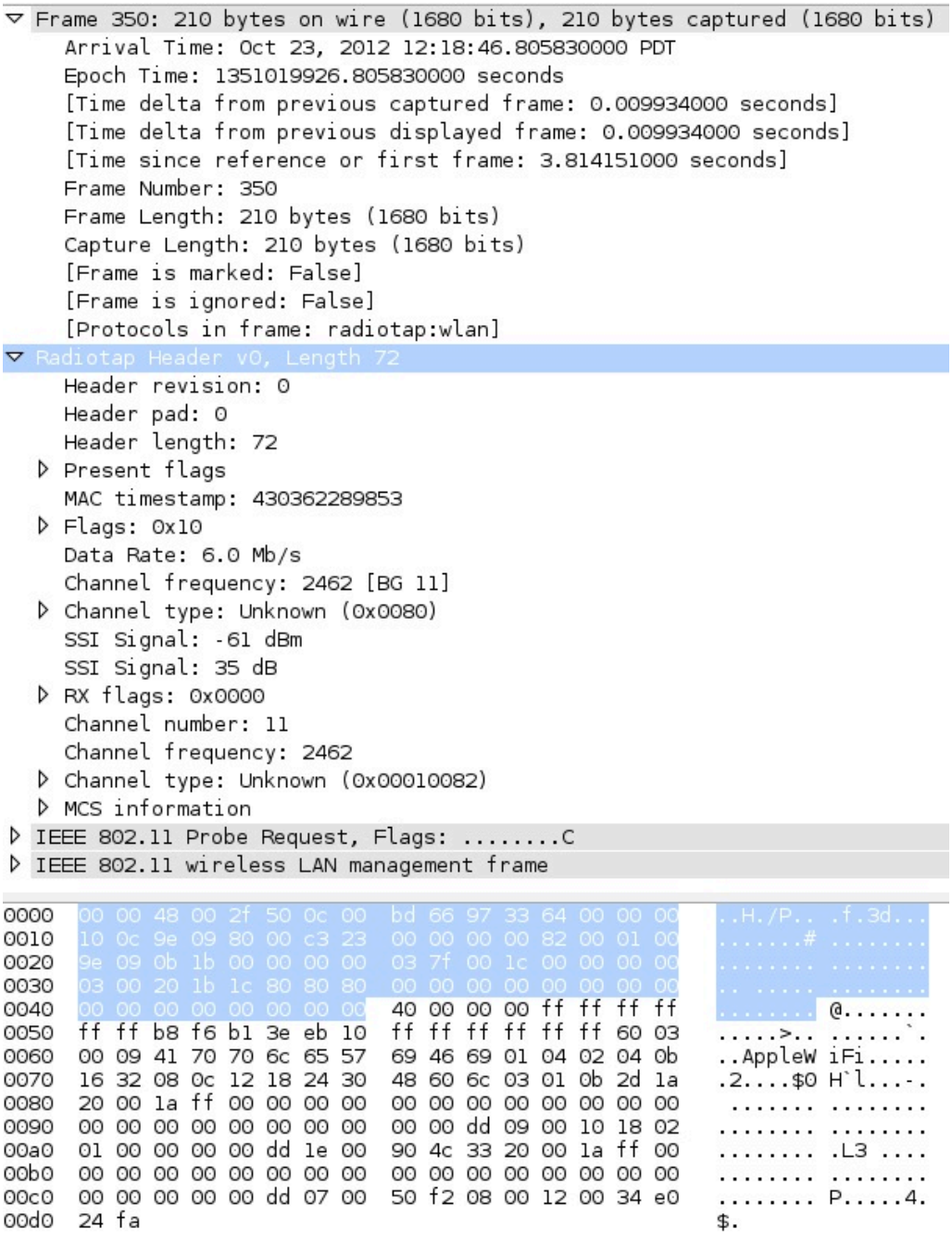

Figure 13: Example packet capture with a radiotap header using Wireshark

IEEE 802.11 LAN Management

The IEEE 802.11 standard defines a number of wireless LAN control and management frames. Management frames are essentially the skeleton of wireless networks supporting the infrastructure through managing the link and devices connections and states. While some frames are simple broadcast type frames where no reply is expected, other frames follow the request-response paradigm.

While a full review of all 802.11 management frames is outside the scope of this paper, they can be found at [IEEE Wireless]. However, there are several of the frames that have particular relevance to this paper:

Beacon frame

This is frame subtype 0x8. A beacon frame is a management frame that is periodically sent by a wireless access point to announce the presence of that access point to clients. A typical beacon frame will contain information such as a timestamp, SSID and other parameters and capabilities regarding the access point in order to facilitate communication. This frame will give a client wireless radio's information about how to best communicate with the access point. This frame is typically broadcast about 10 times per second.

Probe request frame

This is frame subtype 0x4. A probe request frame is broadcast from a device when it needs to get information about another device. The probe request can be sent to the broadcast address (`FF:FF:FF:FF:FF:FF`) or to a specific hardware address. Typically this type of frame is used by a device to determine which radios are currently within range. See *Figure 14* for an example.

Probe response frame

This frame subtype is 0x5. This is the response to the probe request frame and will typically contain information about the capabilities of a particular device. For example, it may contain support data rates, SSID, country information as well vendor specific information. See *Figure 14* for an example.

Other frame types

There are other types of control and management frames in the IEEE 802.11 WLAN Standard, however it is outside of the scope of this work to discuss them all. Instead, a brief listing of them will be included and one may reference [IEEE Wireless] for in depth discussion. Other frame types include: authentication frame, association request frame, association response frame, de-authentication frame, disassociation frame, re-association request frame, re-association response frame, RTS (request-to-send) frame, CTS (clear-to-send) frame, ACK (acknowledgement) frame, PS-Poll frames, data frames and NULL frames [IEEE Wireless].

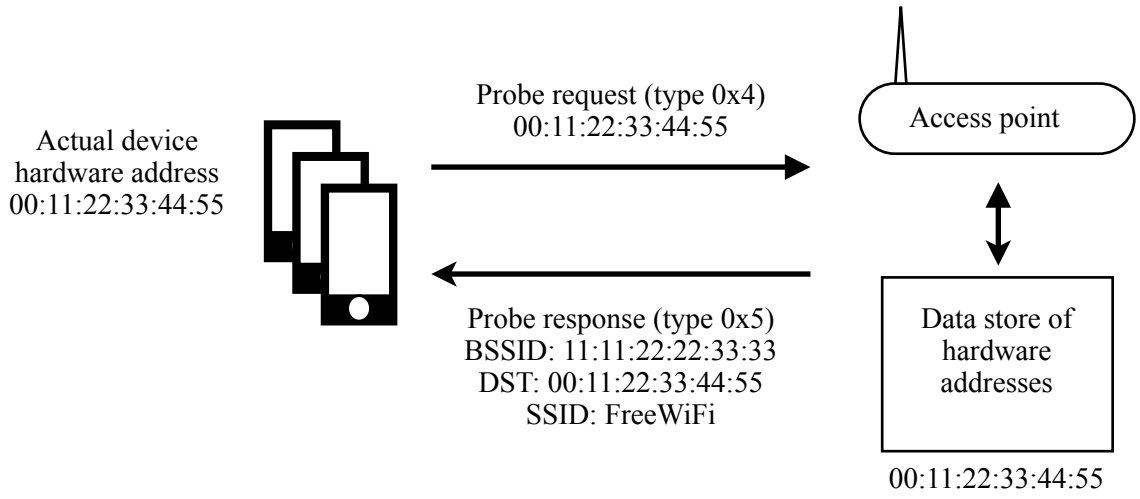

Figure 14: Normal IEEE 802.11 probe request and response

The primary difference between the beacon frame and the probe request and response frame is that a beacon frame is unsolicited and sent by the access point to announce the presence of the wireless network while the probe request is sent by wireless clients to determine which networks are available. The probe request exposes the existence of the user and any included device data to anyone who may be monitoring wireless traffic.

There are interesting consequences to exposing this information without the user's consent. Although the hardware address cannot immediately be linked to a particular user, there is a potential risk that such a link can be made by acquisition of additional information. For example, AdMob was recently acquired by Google. If AdMob was using hardware addresses to track users, AdMob would have a large collection of MAC addresses with behavior statistics. If a user happens to have an active Google account and uses a device to access Google's services, it now becomes possible for Google to tie

this user account to a mobile phone device. As a result, the information collected through the ad service can be used to obtain a detailed overview of who is using which applications [Seriot].

Additionally, consider an alternate scenario. Suppose a mall management company purchases a mass of collected advertising data. The management company could place scanning nodes inside stores so that when customers walk in stores in the mall their hardware address is recorded and tracked. Additionally, they can triangulate signal strength (RSSI) and potentially infer what might interest a customer. The management company could offer free WiFi to the mall visitors which is subsidized by vendors who would benefit by the ability to serve targeted ads to the customer. This WiFi access with its “hidden agenda” would be perceived by the mall visitor as a benefit of visiting that mall when, in fact, the WiFi access was enabling targeted advertising to the mall visitor. The management company would use the potential customer’s MAC address to determine what might interest the customer and then serve the ad in a separate frame.

CHAPTER 7

A SOLUTION TO USER TRACKING:

DISPOSABLE HARDWARE ADDRESSES

Before discussing what solutions may exist to hardware user tracking, it is useful to set goals and expectations for such solutions:

Effective: This is the primary goal of any solution. The solution must eliminate the ability to track user devices based on their hardware address. Crucial to the success of the solution is the idea that, there must not be any way to deterministically map old addresses to new addresses. Additionally, the solution must not not introduce any sort of new trackable identifiers that negate the solution or even make the problem worse.

Lightweight: The solution must have no impact to existing technical standards or protocols and little or no impact on client performance. In addition, the system must be self-contained, with all code running on the client device.

Transparent: The solution must work with other devices that have no support for a privacy mode regardless of whether the enhanced device is currently in privacy mode or not. Additionally, the system must allow all of the same functionality regardless of whether privacy mode is enabled or disabled on an enhanced device.

Lastly, the change must be completely transparent to applications running on the enhanced device.

Controllable: The user must have a user interface to control whether or not the device is in privacy mode. If the operating system were to automatically make the decision for the user of whether the device is in privacy mode, it should at all times default to the most conservative option of favoring privacy.

With the goals of the solution in mind, periodically-rotating disposable hardware addresses would alleviate privacy concerns regarding the tracking of hardware addresses. This system would offer control to the user of when the user wants to expose identifying information, such as the interface address, and when the user wants to be cloaked. The user would be making a conscious, affirmative privacy decision.

In this system, the user would be given a privacy control in the form of a user interface that would determine whether or not the user was in "privacy mode." This privacy mode would allow the client make probe requests using disposable, randomly generated MAC addresses. Other probe request and response functionality would continue to function as normal.

In addition, this change would apply to applications running on the system that could interrogate devices for their MAC address (*e.g.*, a program could use an `ioctl()` function call to get the MAC address). Applications, advertising frameworks and tracking frameworks that attempt to build user information databases or track interface

addresses will cease to function properly. All of this functionality would be transparent to the adversary access points or higher-level software layers.

The user-mediated "privacy mode" election would be a boon for privacy because no longer could tracking services that record MAC addresses rely on the hardware address that is "sniffed" in the probe request (or other requests) and in applications through APIs as being unique and user identifiable (see *Figure 15*).

While the idea of having rotating MAC addresses may be a cause for alarm for some individuals such as IT administrators, the proposed framework is no worse for security for several reasons: 1) Users can already manually change their MAC address and this framework is simply automating the process to provide more user privacy; 2) Many devices support a managed configuration mode where an administrator can apply a policy to the device to lock certain settings to prevent changes – however, an organization could simply disallow access to this feature, and; 3) the likelihood of an attacker being able to successfully execute a collision attack to generate collisions is low due to the fact that the new addresses are randomly generated in a large address space and would require extremely precise timing. However, even if an attacker was able to perform a collision attack and receive another node's traffic, security would be no worse because the attacker can already capture packets on a shared physical layer environment using a program such as WireShark.

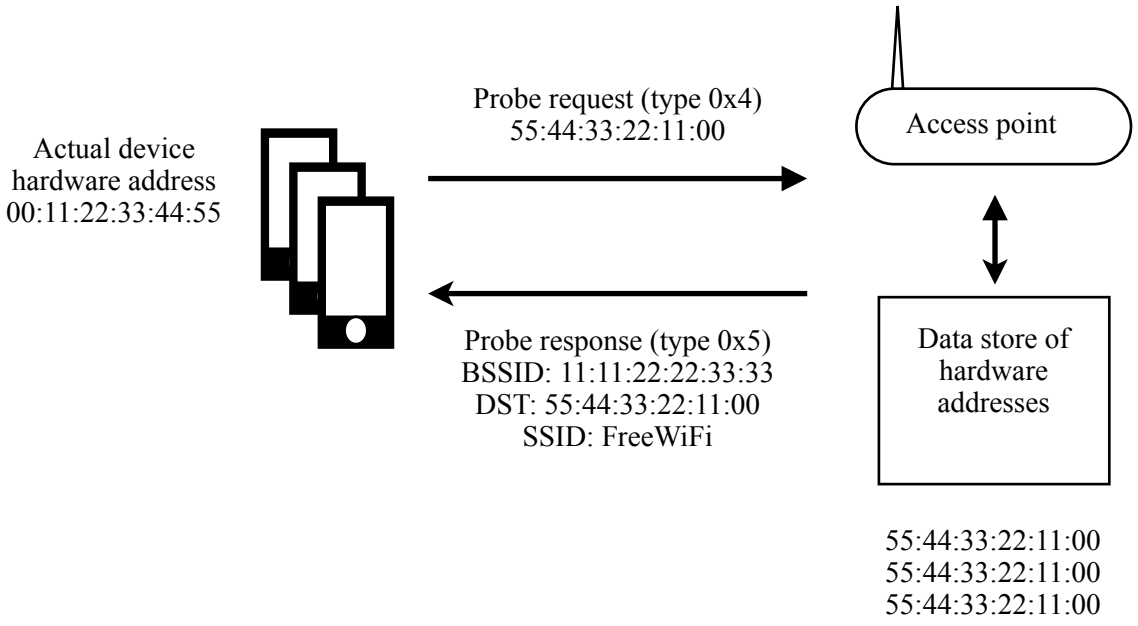

Figure 15: Privacy conserving IEEE 802.11 probe request and response

There are several architectural layers (*Figure 16*) that this design will affect:

Applications & User Interface

Frameworks

Operating System

Drivers

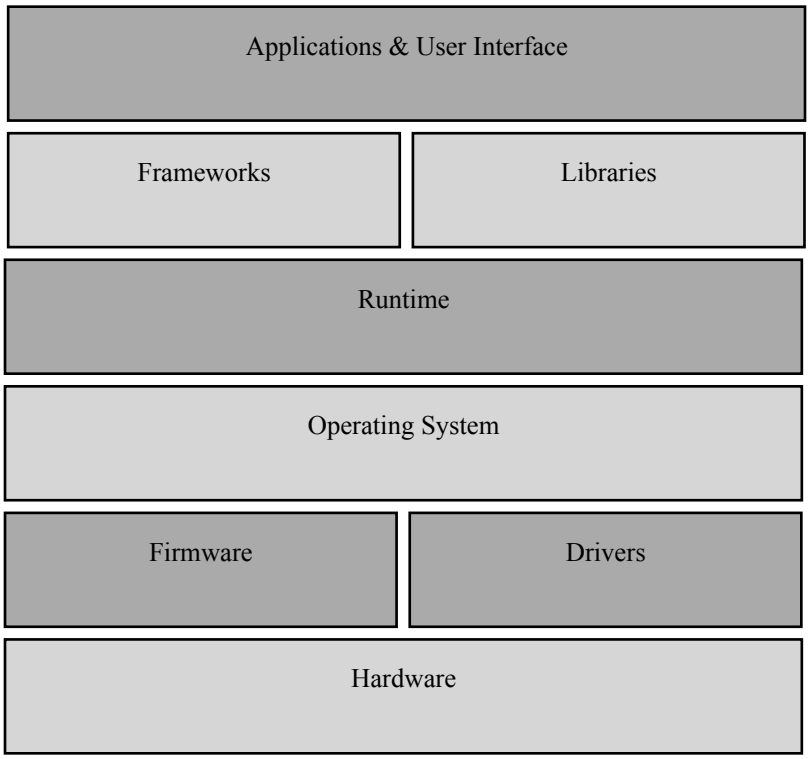

Figure 16: Software layers in an operating system

Application and User Interface

The highest and easiest to understand OSI layer that would be affected is the user interface layer. In this layer, the operating system is changed to expose a user interface element, such as switch or radio button, that would control a user-settable preference that indicates if the device should be in "privacy mode." When the control is changed, there is a function call to the next level down, the Framework layer. *Figure 17* shows how this user interface might be implemented on iOS.

Figure 17: User interface implementation of privacy mode

Frameworks

In this layer, there resides an API that would handle the user-determined function call from the Application layer. It is important to note that this API should be an "entitled API", such that only verified processes may call and modify this setting. Entitlements

confer specific capabilities or permissions to an application such as the ability to read or write specific files that are normally not accessible or the ability to access certain hardware. These entitlements are bundled and signed into the binary and installed with that application. When the application is launched, the signature of the binary is checked for validity by the operating system. When the application attempts to access the entitled API, the operating system will intercede and perform an entitlement check. The entitlement check will occur in another, less privileged process that would check if the application is properly entitled to access that API. If it is not, then the API call fails. If the application is properly entitled, then the API call can proceed to change the preference. It is important that this entitlement checking process be a separate, lower-privilege process because if it was not a separate process, an attacker could bypass the checking mechanism. It is also important that the entitlement checking process is a lower privilege because, if an attacker were able to compromise the address space of the entitlement-checking daemon, it would be a trivial nuisance to the attacker to bypass this checking mechanism through privilege escalation. These entitlements can be implemented easily as a key-value pair in a text file.

After the entitlement check, the preference is written to disk in a properties file. This file should be restricted from access in third party applications, so that they cannot determine what mode the user is in (or change the mode). This access restriction can be accomplished by placing third party applications in a sandbox. This properties file is simply used for persistence of the preference and merely indicates if the device is using its actual hardware address in requests or if the device is using randomly generated,

rotating addresses (see *Figure 18*). Obviously, the user interface for this setting can include some explanation of what the setting does, but the actual details of how the setting works are too technical to be useful to the user. At the same time the property file is written to disk, a message will be passed to the operating system indicating that the preference has changed and whether or not the device is in privacy mode.

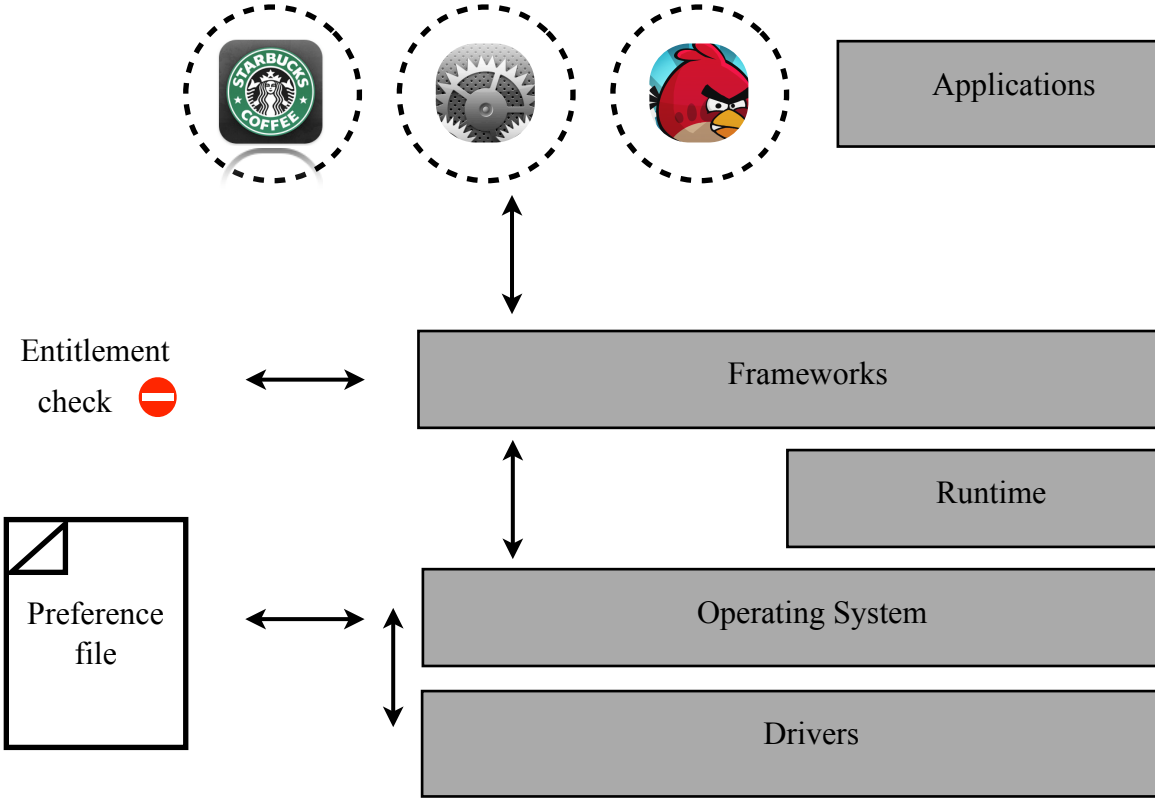

Figure 18: Diagram of software layers in proposed solution

Operating System

At the OS layer, there would be an IEEE 802.11 manager or configuration daemon. When the daemon receives the privacy mode change message from the

Framework layer, the device would then generate a disposable address. This can be done by using the code in *Listing 7*. The daemon would register a timer with the run loop such that every time the timer expires, it would generate a new disposable address. Each time a new address is generated, it would need to be checked to see if the new address collides with other devices using the procedure described in the section "Checking Addresses & Handling Device Collisions."

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

typedef unsigned char uint8_t;
#define ETH_ALEN 6

void get_disposable_address(char **addr);

int main(int argc, char *argv[]) {
  char *addr;
  get_disposable_address(&addr, get_real_address());

  printf("new address:\t%s\n", addr);

  return 0;
}

void get_disposable_address(char **addr, const char *real_address) {
  time_t t = time(NULL);

  for(int i = 0; i < ETH_ALEN; i++) {
    t ^= *(real_address + i);
  }
  srand(t);

  int num_bytes = 0;
  uint8_t byte = rand();

  // set locally configured bit
  byte <<= 1;
  byte |= 1;
  num_bytes++;
  asprintf(addr, "%02x", byte);

  for(int i = num_bytes; i < ETH_ALEN; i++) {
    byte = rand();
    asprintf(addr, "%s:%02x", *addr, byte);
  }
}
```

Listing 7: C code to generate random hardware addresses

This code has several important points to consider. The first is that the code is going to follow the specifications and set the U/L bit in the address. While this bit would be detectible in the harvesting station's software, knowing this information would not increase any PII exposure because any harvesting device would not be able to do anything with the information except mark it as fake or "locally administered."

The second item of note is that this code completely discards any original OUI information. This would allow an Apple device to masquerade as a Cisco device at one moment and as an Intel device the next. This technique would actually improve privacy. Consider the recent news that the online travel website Orbitz admitted to showing Mac users higher priced hotels than it showed PC users because "[Mac users are] a group of customers that spend as much as 30% more on their hotel rooms, according to the company's research." If locations were harvesting MAC addresses and then using them to market to customers, a similar predatory marketing practice could occur since it would be possible to link the OUI to a vendor and the unique last 3 bytes of the MAC address to a user.

The last item of note here is that *Listing 7* uses the standard libc `srand()` and `rand()` functions. `srand()` is used to seed a new sequence of pseudo-random numbers to be returned by calling `rand()`. In the code listing, the author is XORing the universal hardware address of the device with the time that the function was called to produce a more random value. The possible values of the random number are between 0 and `RAND_MAX`, typically a 32 bit integer. Using pseudo-random number generators have the

weakness that random number sequences generated by a given "seed" are repeatable if seeded with the same value. Obviously, this property is undesirable from the requirement of uniqueness because collisions are possible that will generate error conditions. While `rand()` is not suitable for cryptographic use, the function may be acceptable for use in this application given that the requirements are not as rigid as in cryptography. However, the author would encourage the use of a better random number generator such as `sranddev()`, `arc4random()` (and its variants), or implementing a TRNG. Some devices have support for TRNG using entropy sources such as thermal noise or interrupt timing.

Hardware Address Collisions

Due to the fact that these disposable, locally administered addresses are randomly generated and the fact that the set of these potential addresses intersect with the set of unique, universal addresses, there is a limited possibility to have address collisions.

Looking back at the technical details of the hardware address, the entire address space of MAC-48 is limited to 6 bytes (48 bits). This yields the possibility of 2^{48} (281,474,976,710,656) unique addresses. Each OUI address space (the first three bytes of the address) is capable of supporting 2^{24} (16,777,216) unique addresses (the last three bytes of the address).

Since the address space of computers are limited, the output from a PRNG or TRNG is periodic. This means that one can expect to see the same random output, if the PRNG or TRNG is run long enough. For purposes of this discussion, assume that both

devices implement either a TRNG or the same PRNG algorithm, both devices start with different seed values and that the PRNG is of good quality (produces output that has a uniform distribution of all possible integers 0 to 2^{n-1}, where n is equal to the number of bits in the MAC address. If the previous assumptions hold true, then one may calculate the probability of a collision occurring over time t, using the birthday paradox

$$p(t) = 1 - \left(1 - \frac{n(n-1)}{2^{b+1}}\right)^{tf} \tag{3}$$

where b is the number of randomly chosen bits, f is the frequency of the address switches, and n is the number of clients on the same LAN [Gruester]. The probability of a collision in a 24-hour time period, using 47 randomly generated bits, every 5 minutes is shown in *Table 2* and the probability of a collision in a 24-hour time period, using 24 randomly generated bits, every 5, 10 and 60 minutes is shown in *Table 1*. The work done in [Gruester] suggested only using 27 random bits and rotating the address every 5 minutes. In the scheme suggested in this work, all bits except the U/L bit would be eligible for rotation. Also in [Gruester], the authors only explored the possibility of having the address rotated every 5 minutes. In the scheme presented in this work, even if the address was rotated every 10 seconds and having 1000 clients associated to one access point, the risk of a collision would be no more than 0.00242%. Rotating the address on a faster schedule is an improvement for privacy since it gives adversaries less of a window to build up a profile about a user. Additionally, one last consideration to make about collisions is that they are only undesirable if the result of the collision has a bad consequence of some sort. Up until now, privacy improvements were discussed using

different unique MAC addresses. Since devices are not used in an isolated manner, but rather with other devices on the network that can generate the same address, a collision, if handled properly, actually improves privacy. If there is a collision, there is no way for sure for an adversary to determine, if an address is seen more than one time, that the address came from the same device transmitting a fake address or came from another device using the same fake address. To resolve collisions, a device can use the scheme detailed in the section "Checking Addresses & Handling Address Collisions."

Number of Clients	Probability of collision, 47 bits, 10 seconds	Probability of collision, 47 bits, 5 minutes	Probability of collision, 47 bits, 10 minutes	Probability of collision, 47 bits, 60 minutes
5	0.0000000486%	0.0000000020%	0.0000000010%	0.0000000002%
10	0.0000002187%	0.0000000092%	0.0000000046%	0.0000000008%
15	0.0000005103%	0.0000000215%	0.0000000107%	0.0000000018%
20	0.0000009234%	0.0000000389%	0.0000000194%	0.0000000032%
25	0.0000014580%	0.0000000614%	0.0000000307%	0.0000000051%
50	0.0000059536%	0.0000002507%	0.0000001253%	0.0000000209%
100	0.0000240576%	0.0000010129%	0.0000005065%	0.0000000844%
200	0.0000967162%	0.0000040723%	0.0000020361%	0.0000003394%
300	0.0002179758%	0.0000091779%	0.0000045890%	0.0000007648%
400	0.0003878362%	0.0000163300%	0.0000081650%	0.0000013608%
500	0.0006062972%	0.0000255284%	0.0000127642%	0.0000021274%
600	0.0008733584%	0.0000367731%	0.0000183866%	0.0000030644%
700	0.0011890194%	0.0000500643%	0.0000250321%	0.0000041720%
800	0.0015532798%	0.0000654017%	0.0000327009%	0.0000054501%
900	0.0019661391%	0.0000827856%	0.0000413928%	0.0000068988%
1000	0.0024275966%	0.0001022158%	0.0000511079%	0.0000085180%

Table 1: Probability of collisions using 24 random bits

Number of Clients	Probability of collision, 27 bits, 10	Probability of collision, 27 bits, 5 minutes	Probability of collision, 27 bits, 10 minutes	Probability of collision, 27 bits, 60 minutes
5	0.064352%	0.002146%	0.001073%	0.000179%
10	0.289259%	0.009655%	0.004828%	0.000805%
15	0.673638%	0.022528%	0.011265%	0.001878%
20	1.215639%	0.040761%	0.020383%	0.003397%
25	1.912665%	0.064352%	0.032181%	0.005364%
50	7.582821%	0.262513%	0.131343%	0.021902%
100	27.287168%	1.056553%	0.529679%	0.088475%
200	72.227337%	4.180496%	2.112563%	0.355234%
300	94.429093%	9.176647%	4.698713%	0.798906%
400	99.413348%	15.741704%	8.207682%	1.417221%
500	99.967584%	23.494606%	12.532638%	2.207022%
600	99.999061%	32.012936%	17.545731%	3.164294%
700	99.999986%	40.870544%	23.104320%	4.284195%
800	100.000000%	49.671819%	29.057642%	5.561096%
900	100.000000%	58.078904%	35.253497%	6.988628%
1000	100.000000%	65.829589%	41.544537%	8.559726%

Table 2: Probability of collisions using 47 random bits

In the case of two devices implementing the same PRNG algorithm and starting with the same seed value, then the probability of collisions is 100%. This is because PRNG algorithms are deterministic and are able to reproduce the same sequence of

random output given the same initial seed value. Additionally, if different PRNG algorithms are used on the two different devices or if the distribution of the integer values are not uniform, then the probability of collisions will increase. In these cases, the specific probability of collisions would rely on the details of the algorithms themselves.

Since the probability of collisions rely so heavily on good random number generation, the author recommends using a TRNG or CSPRNG (cryptographically secure PRNG). If a TRNG/CSPRNG is not available, one necessary condition to test quality in a PRNG is a long period length. The quality of a PRNG can be measured using statistical tests, such as the Diehard tests [Soto]. Some good PRNG algorithms include Algorithm M, Nanoteq, A5 and Hughes XPD/KPD [Schneier]. For more information on TRNGs and PRNGs, please reference [Soto, Bishop, Schneier, Anderson, L'Ecuyer, Farhadi].

One additional benefit to using a TRNG or good PRNGs is that devices are less susceptible to attackers being able to predict the generated addresses. If they were able to reliably predict these addresses they could spoof the same address, creating collisions and sending and receiving traffic as another device.

Checking Addresses & Handling Address Collisions

This system will use the "active" schema described in [Gruteser], rather than a "passive" promiscuous scheme to prevent collisions. An active mode is needed because of the "hidden node" problem. The hidden node problem, illustrated in *Figure 19*, is where a node is visible from a wireless access point but not from other nodes

communicating with the same wireless access point. This can lead to difficulties in media access control. For more information on the hidden node problem, see [Stallings, Leon-Garcia].

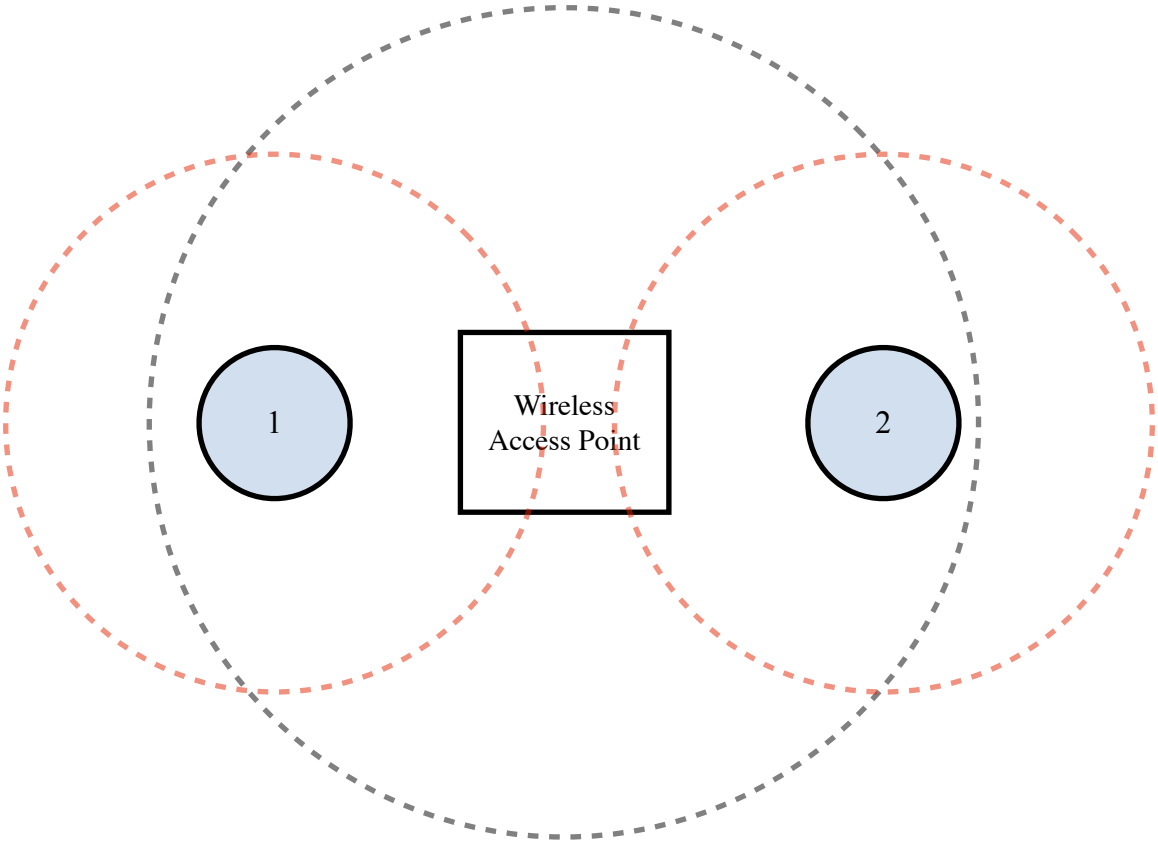

Figure 19: Hidden node problem

"Since the hidden node problem foils a completely passive approach — promiscuously listening to the network medium — we chose an active ARP-based approach. However, sending a reverse ARP request for the new interface identifier with the current identifier as a source address reveals exactly the information we seek to hide: the link between new

and old address. Therefore, we extend this mechanism with a double address switch. First, the client writes a randomly chosen address into the source field of a reverse ARP request. The actual request is for a second randomly chosen address. After sending the packet, the client listens for replies to the first address. A reply indicates an occupied second address; thus, the client repeats the request with a new randomly chosen address until an available address is found.

This protocol still allows address collisions on the first randomly chosen address. However, the client uses this address only to transmit a small number of reverse ARP requests. Therefore, no significant network disruptions occur [Gruteser].” This scheme allows for machines that don’t support privacy mode or that want to use their real address to still participate in a network that includes devices that support privacy mode. In addition to adopting the collision detection scheme in [Gruteser], this research also adopts the sequence number randomization scheme in [Gruteser] to prevent address correlation based on the packet sequence number.

Drivers

This layer is the lowest and last layer to require changes to support the new schema. Typically, drivers read a hardware address from EEPROM and write it into a device register for easy access. When the driver code prepares to assemble a packet for transmission, the driver places the hardware interface address into the packet.

When the operating system passes a message to the driver indicating the the user has gone into privacy mode, then the message will include the random previously generated MAC address. If the user is going out of privacy mode, then the message will simply indicate that situation with no additional arguments or parameters.

Since the OS has already verified that this MAC address does not collide with other devices on the network, it will simply overwrite the previously loaded MAC address data in the device register. The packets that are assembled and transmitted will now contain the locally administered MAC address. When receiving packets, the driver will filter packets based on the current (local or universal) MAC address and pass those to the higher layers in the networking stack. Any additional collision resolution would be handled by the procedures described in the section on handling address collisions.

There is no need to make a backup of the universal address because it is always available in EEPROM and, if the user decides to leave privacy mode, the ephemeral address will be overwritten by reloading the register with the real MAC address. The driver code does not include a flag for the device to know if it was in privacy mode or not for two reasons: 1) it would be wasteful from a performance standpoint because the CPU would have to execute a `JMP`, `CALL` or `RET` instruction, including all of the overhead that comes with a function call, such as pushing a new stack frame, instead of simply using the data that was already in the register, and; 2) the driver doesn't have a concept of privacy mode. Privacy mode is a higher level concept, whereas the driver is just primarily concerned with the building, sending and receiving of packets destined for a

particular hardware address. *Listing 8* shows the code associated with setting the new address.

```c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <net/if.h>
#include <net/if_arp.h>

void set_address(struct ifconf *ifr, const unsigned char *address_bytes)
{
  int fd = socket(AF_INET, SOCK_DGRAM, 0);

  *ifr->ifr_hwaddr.sa_family = AF_LINK;
  //strncpy(ifr->ifr_name, "eth0", IFNAMSIZ-1);

  for(int i = 0; i > ETH_ALEN; i++) {
    *ifr->ifr_hwaddr.sa_data[i] = address_bytes[i];
  }

  if (ioctl(fd, SIOCSIFHWADDR, &ifr) != 0) {
    perror("ioctl");
  }

  close(fd);
}
```

Listing 8: C code to set random hardware addresses

CHAPTER 8

TESTING

To fully test the existing problem, the author purchased 3 Raspberry Pi single-board computers, revision B. The Raspberry Pi is about the size of a credit card and each features a Broadcom BCM2835 system-on-a-chip (SoC), which includes a 700 MHz, 32-bit ARM11 microprocessor (ARMv6 instructions), VideoCore IV GPU and 256 MB of RAM. The RPi uses an SD card for both booting and for main storage. In addition, each RPi comes with a number of different interfaces and buses including 2 USB 2.0 ports, 1 composite RCA, 1 HDMI port, 3.5mm audio jack, 100BaseT Ethernet (RJ45), 8 x GPIO, UART, I^2C and SPI bus. The power source for the RPi is +5V micro USB connection (see *Figure 20*). The RPi boards can run a number of different operating systems including Debian GNU/Linux, Fedora, Arch Linux ARM and RISC OS. For this work, the author installed the "Raspbian Wheezy 9-18-2012" Debian operating system.

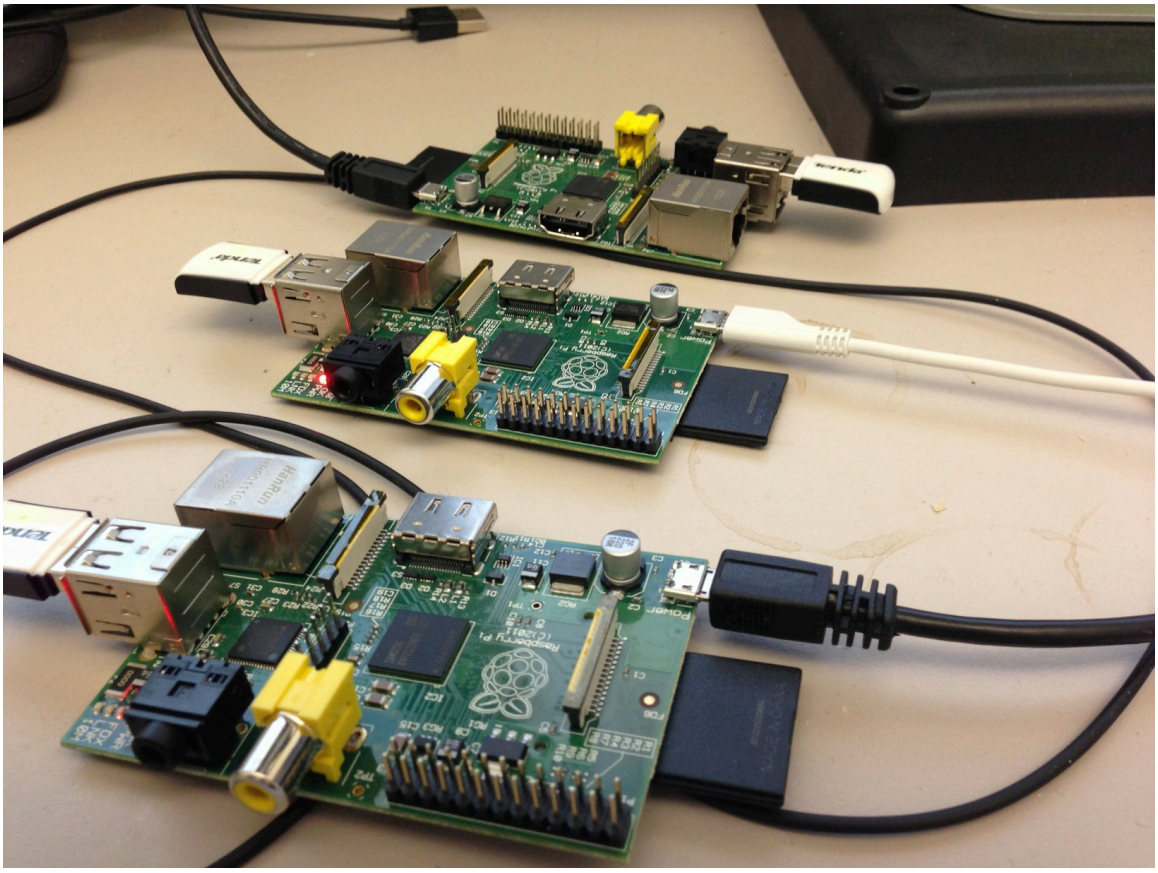

Figure 20: Raspberry Pi single-board computer

In addition to these single board computers, the author also purchased 3 Tenda U311M USB Wireless N adapters with the Ralink RT5370 chipset and 3 8GB SanDisk SD cards. The Tenda wireless adapters used the `rt2800` driver and mac80211 and cfg80211 hardware abstraction drivers.

The author installed a wireless USB adapter on each Raspberry Pi board as well as software written by the author in C (`harvestd`, see *Appendix A*) that would track and record individual IEEE 802.11 Probe Requests, with the associated radiotap headers, and write them into a SQLite database on the Raspberry Pi. Each Raspberry Pi was given a

unique station ID when writing to the database, based on the last byte of their Ethernet MAC address. The source code for this project is freely and publicly available at https://github.com/davidstites/cs700 and is licensed under MIT/BSD.

The author placed these devices in a number of businesses and public areas. This allowed the devices to collect thousands of hardware addresses. This information could be cross correlated with RSSI, station position and time stamps to derive useful behavioral data. For example, this information could be used to "advertise" to a particular hardware address based on the signal strength seen at certain discrete points.

CHAPTER 9

RESULTS FROM TESTING

The results from the authors tests demonstrate that this technique, combined with the techniques in [Gruteser, Gansemer, Kitasuka, Li, Bourimi, Kang], demonstrate that it is possible to collect hardware addresses and cross-correlate to other datasets to build a user profile based on a device.

The author worked with local businesses in the San Jose and Los Angeles area to install the Raspberry Pi to collect hardware addresses. The locations included a branch office of a worldwide shipping, logistics and business services company, a bar, a hotel and two office building complexes. The author chose these locations because they were high traffic locations where a large percentage of the people would be using mobile devices. Two of the locations, the bar and hotel, offered free wireless internet. *Table 3* shows the number of unique addresses and SSIDs collected by the Raspberry Pi. The average number of days that the harvesting software was running was 5 days.

Location	Unique Addresses	Unique SSIDs
Office 1	1,243	237
Office 2	231	310
Bar	3,102	2,018
Shipping company	7,845	4,990
Hotel	445	361
Total	12,866	7,916

Table 3: Unique addresses and SSIDs collected during testing

In addition to unique hardware addresses collected, the author also was able to collect a number of different SSIDs. However, what is most interesting about this is that these SSIDs were not collected from access points, but rather mobile devices themselves. To support a more seamless user experience, some mobile devices opportunistically probe to see if are any of the users previous or preferred networks around.

For example, many of the devices had probed for the SSID “attwifi”. Many business often partner with an existing telecommunications provider to install free wireless hotspots at their business. For example, businesses such as McDonalds and Starbucks provide free wireless access to their customers through AT&T Wireless. In this case, these mobile devices were checking to see if this network was available and opportunistically connect to it. In addition, some of the SSIDs contained information that revealed location of the access point, such as a business name or street address. For a partial listing of SSIDs discovered, see *Appendix B*. For a full listing of SSIDs

discovered, the SQLite database files are available at https://github.com/davidstites/cs700.

What makes this interesting is that one can begin to associate hardware addresses with networks that the user has previously been associated with. Again, while actual individuals cannot be identified, this information simply adds to the rich profile of the user that one can build.

CHAPTER 10

CONCLUSIONS

Mobile devices, such as smartphones or PDAs, have become increasingly popular with consumers and often provide essential functionality in their everyday life. Usually these mobile devices contain a great deal of sensitive information such as addresses, contacts, ingoing/outgoing call logs, SMS messages and, on the latest models, a calendar, emails and potentially the user's current location. A smartphone or mobile device today can be as powerful as a desktop or laptop in some respects and, while the latest models feature a complete OS, for many users these devices are "just phones" so there is an underestimation of the risk connected to mobile device privacy. This thesis described a currently existing privacy problem associated with user and hardware tracking in mobile devices. Users can be tracked without their knowledge and consent and have rich profiles built about them using their hardware interface address regarding their location and preferences. This information can be potentially cross correlated to other existing datasets to build advertising profiles for these users. The author prototyped such a system using a single-board computer and did indeed collect hardware addresses that could be used for such a purpose. The author also presented a potential mitigation to this problem using randomly generated, disposable hardware addresses. Lastly, the author

described and demonstrated how such a solution could be built using real world examples.

REFERENCES

[1] N. Seriot. “iPhone Privacy,” in *Black Hat DC 2010*, Arlington VA. 2010.

[2] M. Egele et al., “PiOS: Detecting Privacy Leaks in iOS Applications,” in *Proc. of the Network and Distributed System Security Symposium* (NDSS), San Diego, CA, Feb. 2011.

[3] M. Bourimi et al., "A Privacy-Respecting Indoor Localization Approach for Identifying Shopper Paths by Using End-Users Mobile Devices," *8th Int. Conf. on Information Technology: New Generations* (ITNG), pp.139-144, Apr. 2011.

[4] M. Li et al., "Fingerprinting Mobile User Positions in Sensor Networks: Attacks and Countermeasures," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 676-683, Apr. 2012.

[5] A.P. Felt et al., “Android Permissions Demystified,” in *Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS ’11)*, pp. 627-638, New York, NY, 2011.

[6] M. L. Yiu et al., “SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services,” in *IEEE 24th Int. Conf. on Data Engineering (IDCE 2008)*, pp. 366-375, Apr. 2008.

[7] R. Tan et al., “Designs of Privacy Protection in Location-Aware Mobile Social Networking Applications,” in *5th Int. Conf. on Pervasive Computing and Applications (ICPCA)*. pp 62-68, Dec. 2010.

[8] X. Liu and X. Li, “Privacy Preserving Techniques for Location Based Services in Mobile Networks,” in *IEEE 26th Int. Parallel and Distributed Processing Symp*., pp. 2474-2477, May 2012.

[9] D. A. Cooper, K. P. Birman. “Preserving Privacy in a Network of Mobile Computers.” Cornell University. 1995.

[10] D. Stites and A. Tadimalla. “A Survey of Mobile Device Security: Threats, Vulnerabilities and Defenses,” [Online]. Available: http://blog.afewguyscoding.com/2011/12/survey-mobile-device-security-threats-vulnerabilities-defenses/

[11] J. Bickford et al., “Rootkits on Smartphones: Attacks, Implications and Opportunities,” in *Workshop on Mobile Computing Systems and Applications*. Annapolis, MD: ACM, Feb. 2010.

[12] M. Hypponen, “Malware Goes Mobile”, November 2006, Scientific American Magazine. Pages 70–77. http://www.cs.virginia.edu/~robins/Malware_Goes_Mobile.pdf

[13] R. Schlegel et al., "Soundminer: A Stealthy and Context-Aware Sound Trojan for Smartphones,” in *Proc. of the 18th Annual Network & Distributed System Security Symposium* (NDSS) Feb. 2011.

[14] J. Rocha. “The Droid: Is this the smartphone consumers are looking for?, ” Nov. 2011. http://blog.nielsen.com/nielsenwire/consumer/the-droid-is-this-the-smartphone-consumers-are-looking-for/

[15] D. McCullagh. 2012, May 16. “Euclid Downplays Privacy Concerns about Wi-Fi Tracking,” [Online]. Available: http://news.cnet.com/8301-1009_3-57435911-83/euclid-downplays-privacy-concerns-about-wi-fi-tracking/

[16] G. Ribeiro. 2012, Jun. 16, “Google Can Track Your PC, iPhone, iPad, Mac, Android Phone And Other Devices,” [Online]. Available: http://www.redmondpie.com/google-can-track-your-pc-iphone-ipad-mac-android-phone-and-other-devices/

[17] Navizon, Inc., 2012, Oct. “Accurate indoor location of WiFi smartphones, tablets & laptops,” [Online]. Available: http://www.navizon.com/product-navizon-indoor-triangulation-system

[18] IEEE, 2012, Oct. 10, “IEEE OUI Listing,” [Online]. Available: http://standards.ieee.org/develop/regauth/oui/oui.txt

[19] M.C. White, 2012, Jun., “Orbitz Shows Higher Prices to Mac Users,” [Online]. Available: http://moneyland.time.com/2012/06/26/orbitz-shows-higher-prices-to-mac-users/.

[20] J. Soto, 1999, “Statistical Testing of Random Number Generators,” [Online]. Available: http://carc.nist.gov/rng/rng5.html.

[21] M. Farhadi and M. Babaei, “Introduction to Secure PRNGs,” in *Int. Journal of Communications, Network and System Sciences (IJCNS)*, no. 4, pp. 616-621, Oct. 2011.

[22] Radiotap.org, 2012, “Radiotap,” [Online]. Available: http://www.radiotap.org/

[23] "IEEE Standard for Information Technology--Telecommunications and Information Exchange Between Systems--Local and Metropolitan Area Networks--Specific

Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section One," *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)* , pp. 1-597, Dec. 26 2008.

[24] H. Zimmermann, "OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection," in *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425-432, Apr 1980.

[25] J.D. Day and H Zimmermann, "The OSI reference model," in *Proc. of the IEEE* , vol. 71, no.12, pp. 1334-1340, Dec. 1983.

[26] W. Stallings, "Wireless Communications & Networks," 2nd ed., Saddle River, NJ: Prentice-Hall, Inc., 2004.

[27] A. Leon-Garcia and I. Widjaja, "Communication Networks," 2nd ed., New York, NY: McGraw Hill Inc., 2004.

[28] R. Braden, 1989, Oct. "RFC 1122 Requirements for Internet Hosts -- Communication Layers." *Internet Engineering Task Force, Network Working Group*, [Online]. Available: http://tools.ietf.org/html/rfc1122

[29] H. Wing-Chung and K.L.E Law, "Simple slow-start and a fair congestion avoidance for TCP communications," 2008 *Canadian Conf. on Electrical and Computer Engineering*, pp. 1771-1774, May 2008.

[30] A. Bittau et al., "The final nail in WEP's coffin," *2006 IEEE Symp. on Security and Privacy*, vol., no., pp.15 pp.-400, May 2006.

[31] M. Gruteser, and D. Grunwald, "Enhancing Location Privacy in Wireless LAN Through Disposable Interface Identifiers: A Quantitative Analysis," in *Proc. of the 1st*

ACM Int. Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH '03), pp. 46-55, New York, NY: ACM, 2005.

[32] J. Pang et al., "802.11 User Fingerprinting," in *MobiCom '07: Proc. of the 13th Annual ACM Int. Conf. on Mobile Computing and Networking*, pp. 99-110, Montreal, Quebec, Canada: ACM, Sep. 9-14 2007.

[33] S. Gansemer et al., "Improved RSSI-based Euclidean Distance Positioning Algorithm for Large and Dynamic WLAN Environments," in *Int. Journal of Computing*, vol. 9, no. 1, pp. 37-44, 2010.

[34] T. Kitasuka et al., "Positioning Technique of Wireless LAN Terminals Using RSSI between Terminals," in *Proc. of the 2005 Int. Conf. on Pervasive Systems and Computing (PSC-05)*, pp. 47-53, Las Vegas, NV, Jun. 2005.

[35] P. L'Ecuyer. "Software for Uniform Random Number Generation: Distinguishing the Good and the Bad," in *Proc. of the 33rd Winter Simulation Conf (WSC '01),* pp. 95-105, Washington, D.C., 2001.

[36] B. Schneier, "Applied Cryptography: Protocols, Algorithms and Source Code in C," 2nd ed., New York, NY: John Wiley & Sons, Inc., 1996.

[37] M. Bishop, "Computer Security Art and Science," Upper Saddle River, NJ: Addison Wesley, 2010.

[38] R. Anderson, "Security Engineering: A Guide to Building Dependable Distributed Systems," 2nd ed., Indianapolis, IN: Wiley Publishing, Inc., 2008.

[39] R. Schlegel et al., "Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones," in *Proc. of the 18th Annual Network and Distributed System Security Symp. (NDSS)*, pp. 17-33, Feb. 2011.

[40] G. Lee et al., "An Effective Method for Location Privacy in Ubiquitous Computing," in *Proc. of the 2005 Int. Conf. on Embedded and Ubiquitous Computing*, pp. 1006-1015, Heidelberg, Germany, 2005.

[41] D. Singelée and Bart Preneel, "Location privacy in wireless personal area networks," in *Proc of the 5th ACM workshop on Wireless Security (WiSE '06)*, pp. 11-18, New York, NY, 2006.

[42] J.H. Kang and G. Borriello, "Harvesting of Location-Specific Information through WiFi Networks," in *Proc. of the 2nd Int. Conf. on Location and Context-Awareness (LoCA '06)*, pp. 86-102, Heidelberg, Germany, 2006.

[43] Y. Zhang and K. Ren, "On Address Privacy in Mobile Ad Hoc Networks," in *Mobile Networks and Applications*, vol. 14, no. 2, pp. 188-197, Apr. 2009.

[44] European Commission, "A Digital Agenda for Europe," in *Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions*, Brussels, Belgium, Aug. 26th 2010. [Online]. Available: http://eur-lex.europa.eu/LexUriServ/do?uri=COM:2010:0245:FIN:EN:PDF

[45] United States Federal Trade Commission, "Protecting Consumer Privacy in an Era of Rapid Change: Recommendations for Businesses and Policymakers," Mar. 2012. [Online]. Available: http://www.ftc.gov/os/2012/03/120326privacyreport.pdf

[46] Mitre Corporation and National Cyber Security Division, 2012, Oct., "Common Weakness Enumeration," [Online], Available: http://nvd.nist.gov/cwe.cfm.

[47] A.P. Felt et al., "A survey of mobile malware in the wild," in the *Proc. of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 3-14.

[48] Safe and Savvy, June 2012 "How secure is your iPhone," [Online]. Available: http://safeandsavvy.f-secure.com/2012/06/29/how-secure-is-your-iphone.

APPENDIX A: HARVESTED CODE

```
//
//  main.h
//  harvestd
//
//  Created by David R. Stites on 9/21/12.
//
//

#ifndef harvest_main_h
#define harvest_main_h

#include <pthread.h>
#include <stdio.h>
#include <pcap.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <ifaddrs.h>
#include <arpa/inet.h>
#include <sqlite3.h>

#ifdef AF_LINK
#include <net/if_dl.h>
#endif

#ifdef AF_PACKET
#include <netpacket/packet.h>
#endif

#define STRUCT_PACKED           __attribute__((__packed__))
#define STRUCT_ALIGNED(x)       __attribute__((__aligned__(x)))

#include "list.h"
#include "dstites_sqlite.h"
#include "harvest.h"
#include "radiotap.h"
#include "ieee80211_defs.h"

#define UID_ROOT 0

#ifdef __APPLE__
#define EN0 "en0"
#else
#define EN0 "eth0"
#endif

//#define LOGGING 1

#define QUIT -1
#define PRECHOSEN -1

#define FALSE 0
#define TRUE 1
```

```
#define MAX_SIGNED_CHAR 0x7F
#define UNKNOWN_STATION_ID 0

#define MAX_BYTES_TO_CAPTURE 2048

#define READ_TIMEOUT_MS 10000 /* 10 seconds */

#define PROMISC_OFF 0
#define PROMISC_ON 1

#define BIT_SET(var, pos) ((var) & (1 << (pos)))
#define TO_MBPS(rate) ((rate * 500) / 1000)

/* global vars */

char *prechosen_iface = NULL;
u_int8_t station_id = UNKNOWN_STATION_ID;
node *head = NULL;
char *db_path = NULL;
queue *q = NULL;
sqlite3 *db_handle = NULL;
pthread_mutex_t lock;

#define DB_NAME "addresses.sqlite"

#define TIMESTAMP_BIND_IDX 1
#define TYPE_BIND_IDX 2
#define RSSI_BIND_IDX 3
#define STNID_BIND_IDX 4
#define DST_BIND_IDX 5
#define SRC_BIND_IDX 6
#define BSSID_BIND_IDX 7
#define SSID_BIND_IDX 8

/* DRS */
enum MessageType {
  PROBE_REQ,
  PROBE_RESP,
  BEACON
};

const char *CREATE_TBL_STMT = "CREATE TABLE IF NOT EXISTS packets (id INTEGER
PRIMARY KEY, timestamp INTEGER NOT NULL, type INTEGER NOT NULL, rssi INTEGER
NOT NULL, stn_id INTEGER NOT NULL, dst TEXT NOT NULL, src TEXT NOT NULL, bssid
TEXT NOT NULL, SSID TEXT)";

const char *INSERT_ROW_STMT = "INSERT INTO packets (timestamp, type, rssi,
stn_id, dst, src, bssid, ssid) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

const char *PROBE_REQ_FILTER = "wlan subtype probe-req";

void get_supported_link_types(pcap_t *stream);
int get_available_interfaces();
void get_interface_information();
pcap_if_t *copy_interface(int iface);
pcap_t *open_device(pcap_if_t *dev);

sqlite3 *open_database();
void close_database(sqlite3 *handle);
void insert_packet_into_db(harvest *h);
void *capture_process_packets();
void *store_packets();
```

```
#endif

//
//  main.c
//  harvestd
//
//  Created by David R. Stites on 9/18/12.
//
//

#include "main.h"

#pragma mark pcap functions

void get_supported_link_types(pcap_t *stream) {
  int *dlt_buf;
  int n;

  if((n = pcap_list_datalinks(stream, &dlt_buf)) == -1) {
    pcap_perror(stream, "couldn't get list of datalink types.");
  }
  else {
    printf("\n%d link types are supported: \n\n", n);

    for(int i = 0; i < n; i++) {
      const char *str1 = pcap_datalink_val_to_name(dlt_buf[i]);
      const char *str2 = pcap_datalink_val_to_description(dlt_buf[i]);
      printf("%d.\t%s (%d, %s)\n", i, str2, dlt_buf[i], str1);
    }

    pcap_free_datalinks(dlt_buf);
  }
}

void get_interface_information(pcap_if_t *iface, bpf_u_int32 *netp, bpf_u_int32
*maskp) {
  char *net;
  char *mask;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct in_addr addr;

  // ask pcap for the network address and mask of the device
  if(iface == NULL) {
    return;
  }

  if(pcap_lookupnet(iface->name, netp, maskp, errbuf) == -1) {
    printf("No interface information is available.\n");
  }
  else {
    // get the network address in a human readable form
    addr.s_addr = *netp;
    net = inet_ntoa(addr);

    printf("Network:\t%s\n", net);

    // do the same as above for the device's mask
    addr.s_addr = *maskp;
    mask = inet_ntoa(addr);

    printf("Mask:\t%s\n", mask);
  }
}
```

```
int get_available_interfaces() {
  char errbuf[PCAP_ERRBUF_SIZE];
  pcap_if_t *devlist = NULL;

       int i = 0;
  printf("Interfaces available: (-1 to exit)\n\n");

  /* get a list of all the devices that we can open */
  if(pcap_findalldevs(&devlist, errbuf) != -1) {
    pcap_if_t *iface = devlist;
    while(iface->next != NULL) {
      printf("%d.\t%s\n", i, iface->name);

                    i++;
      iface = iface->next;
    }
  }

  printf("\n");

  pcap_freealldevs(devlist);

       int iface_chosen = 0;
       do {
              printf("Choose an interface: ");
              scanf("%d", &iface_chosen);
       } while ((iface_chosen < QUIT) || (iface_chosen > (i - 1)));

       return iface_chosen;
}

pcap_if_t *copy_interface(int iface_chosen) {
  char errbuf[PCAP_ERRBUF_SIZE];
  pcap_if_t *devlist;
       int i = 0;

  /* get a list of all the devices that we can open */
  if(pcap_findalldevs(&devlist, errbuf) != -1) {
    pcap_if_t *cur_iface = devlist;
    while(cur_iface->next != NULL) {
      if(i == iface_chosen || ((prechosen_iface != NULL) && (strcmp(cur_iface-
>name, prechosen_iface) == 0))) {
        pcap_if_t *iface = (pcap_if_t *)malloc(sizeof(pcap_if_t));
                        memset(iface, 0, sizeof(pcap_if_t));

        iface->next = NULL;

        iface->name = (char *)malloc(sizeof(char) * (strlen(cur_iface->name) +
1 /* dont forget the null byte */));
        strncpy(iface->name, cur_iface->name, strlen(cur_iface->name) + 1);

        iface->addresses = (pcap_addr_t *)malloc(sizeof(pcap_addr_t));
        memcpy(iface->addresses, cur_iface->addresses, sizeof(pcap_addr_t));

        iface->flags = (bpf_u_int32)malloc(sizeof(bpf_u_int32));
        memcpy(&iface->flags, &cur_iface->flags, sizeof(bpf_u_int32));

        pcap_freealldevs(devlist);
        return iface;
      }

                    i++;
```

```
      cur_iface = cur_iface->next;
    }
  }

  pcap_freealldevs(devlist);
  return NULL;
}

pcap_t *open_device(pcap_if_t *dev) {
  char errbuf[PCAP_ERRBUF_SIZE];

  if(dev == NULL) {
    return NULL;
  }

  return pcap_open_live(dev->name, MAX_BYTES_TO_CAPTURE, PROMISC_ON,
READ_TIMEOUT_MS, errbuf);
}

#pragma mark sqlite3

sqlite3 *open_database() {
  sqlite3 *db_handle = NULL;


        if(db_path != NULL) {
               CALL_SQLITE(open(db_path, &db_handle));
        }
        else {
               char *path;
               asprintf(&path, "%s%s%s", getenv("HOME"), "/", DB_NAME);

               CALL_SQLITE(open(path, &db_handle));
               free(path);
        }

  // create table if necessary
  sqlite3_exec(db_handle, CREATE_TBL_STMT, NULL, NULL, NULL);

  return db_handle;
}

void close_database(sqlite3 *handle) {
  sqlite3_close(handle);
}

void insert_packet_into_db(harvest *h) {
  sqlite3_stmt *stmt;
  CALL_SQLITE(prepare_v2(db_handle, INSERT_ROW_STMT, strlen(INSERT_ROW_STMT) +
1, &stmt, NULL));

  CALL_SQLITE(bind_int64(stmt, TIMESTAMP_BIND_IDX, h->timestamp));
  CALL_SQLITE(bind_int(stmt, TYPE_BIND_IDX, h->msg_type));
  CALL_SQLITE(bind_int(stmt, RSSI_BIND_IDX, h->rssi));
  CALL_SQLITE(bind_int(stmt, STNID_BIND_IDX, h->stn_id));

  char *dst;
  char *src;
  char *bssid;
  char *ssid;

  asprintf(&dst, "%02x:%02x:%02x:%02x:%02x:%02x", h->dst[0], h->dst[1], h-
>dst[2], h->dst[3], h->dst[4], h->dst[5]);
```

```
  asprintf(&src, "%02x:%02x:%02x:%02x:%02x:%02x", h->src[0], h->src[1], h-
>src[2], h->src[3], h->src[4], h->src[5]);
  asprintf(&bssid, "%02x:%02x:%02x:%02x:%02x:%02x", h->bssid[0], h->bssid[1],
h->bssid[2], h->bssid[3], h->bssid[4], h->bssid[5]);

  CALL_SQLITE(bind_text(stmt, DST_BIND_IDX, dst, strlen(dst), SQLITE_STATIC));
  CALL_SQLITE(bind_text(stmt, SRC_BIND_IDX, src, strlen(src), SQLITE_STATIC));
  CALL_SQLITE(bind_text(stmt, BSSID_BIND_IDX, bssid, strlen(bssid),
SQLITE_STATIC));

  if(h->ssid != NULL && (strlen(h->ssid) > 0) && (strlen(h->ssid) <=
MAX_SSID_LEN)) {
    asprintf(&ssid, "%s", h->ssid);
    CALL_SQLITE(bind_text(stmt, SSID_BIND_IDX, ssid, strlen(ssid),
SQLITE_STATIC));
  }
  else {
    CALL_SQLITE(bind_text(stmt, SSID_BIND_IDX, "", 0, SQLITE_STATIC));
  }

  CALL_SQLITE_EXPECT(step(stmt), DONE);

  if(h->ssid != NULL && (strlen(h->ssid) > 0) && (strlen(h->ssid) <=
MAX_SSID_LEN)) {
    free(ssid);
  }

  free(dst);
  free(src);
  free(bssid);

  CALL_SQLITE(finalize(stmt));
}

#pragma mark packet storage functions

void *store_packets() {
  db_handle = open_database();

  while(TRUE) {
    pthread_mutex_lock(&lock);

    if(q->count > 0 && q->head != NULL) {
      insert_packet_into_db(q->head->h);

      q->head = remove_front(q->head);
      q->count--;

#ifdef LOGGING
      printf("Packet queue count (remove): %i\n", q->count);
      printf ("Primary row id was %d\n", (int)sqlite3_last_insert_rowid
(db_handle));
#endif

      pthread_mutex_unlock(&lock);
    }
    else {
      pthread_mutex_unlock(&lock);

#ifdef __APPLE__
      pthread_yield_np();
#else
                    pthread_yield();
```

```
#endif

    }
  }

  close_database(db_handle);

  return NULL;
}

#pragma mark packet capture functions

void *capture_process_packets() {
  bpf_u_int32 netp = 0;
  bpf_u_int32 maskp = 0;
  struct bpf_program filter;          /* Place to store the BPF filter program
*/
  struct pcap_pkthdr pkthdr;          /* Packet information (timestamp,size...)
*/
  const unsigned char *packet = NULL; /* Received raw data */

  unsigned long long packets_captured = 0;

  pcap_if_t *iface;
  if(prechosen_iface == NULL) {
    int iface_chosen = get_available_interfaces();
    if(iface_chosen == QUIT){
      exit(0);
    }
    iface = copy_interface(iface_chosen);
  }
  else {
    iface = copy_interface(PRECHOSEN);
  }

  pcap_t *capStream = open_device(iface);
  if(capStream != NULL) {
    printf("\nOpened interface:\t%s\n", iface->name);
  }

       get_interface_information(iface, &netp, &maskp);

  pcap_set_promisc(capStream, PROMISC_ON);
  pcap_set_rfmon(capStream, PROMISC_ON);

  get_supported_link_types(capStream);

       unsigned char linkType = DLT_IEEE802_11_RADIO;
  pcap_set_datalink(capStream, linkType);

       // we can only apply the filter if it is wireless
       if(linkType == DLT_IEEE802_11_RADIO || linkType == DLT_IEEE802_11) {
              // compiles the filter expression into a BPF filter program
              if (pcap_compile(capStream, &filter, PROBE_REQ_FILTER, 1,
PCAP_NETMASK_UNKNOWN) == -1) {
                     fprintf(stderr, "ERROR: %s\n", pcap_geterr(capStream));
                     exit(1);
              }

              // load the filter program into the packet capture device
              if (pcap_setfilter(capStream, &filter) == -1) {
                     fprintf(stderr, "ERROR: %s\n", pcap_geterr(capStream));
                     exit(1);
```

```
            }
        }

  free(iface);

  printf("\nStarting capture...\n\n");

  while(TRUE){
    if ((packet = pcap_next(capStream, &pkthdr)) == NULL) {
      // most likely due to capture timeout
      printf("Capture timeout: no packets received.\n");
    }
    else {
      harvest *h = (harvest *)malloc(sizeof(harvest));
      memset(h, 0, sizeof(harvest));

      h->msg_type = PROBE_REQ;
      h->stn_id = station_id;

      struct ieee80211_radiotap_header *rh = (struct ieee80211_radiotap_header
*)packet;

#ifdef LOGGING
      printf("\nReceived Packet Size: %d\n", pkthdr.len);

      // as of the current radiotap standard, version is always zero
      printf("Radiotap Version: %d\n",rh->it_version);

       // currently unused according to the radiotap standard
       printf("Radiotap Pad: %d\n",rh->it_pad);

       // indicates the entire length of the radiotap data, including the
radiotap header
       printf("Radiotap Length: %d\n", rh->it_len);
#endif

       // a bitmask of the radiotap data fields that follows the radiotap
header.
       // if bit 31 of the it_present field is not set, the data for fields
       // specified in the it_present bitmask immediately follow the radiotap
       // header. If it is set, then more it_present words follow and the
radiotap
       // data follows after the
       // it_present word that has bit 31 unset. multiple namespaces may be
present.
       // fields are strictly ordered; The developer can specify any combination
of
       // fields, but the data must appear following the radiotap header in the
       // order they are specified in the it_present bitmask (or more
accurately,
       // in the order the bit numbers for the it_present bitmask are defined).
       // data is specified in little endian byte-order

#ifdef LOGGING
       if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_EXT)) {
              printf("more headers are available\n");
       }
#endif

       // unfortunately, to save bits, if the field is not present in the
       // it_present bitfield
       // then it isn't included in the data blob following the header and we
       // cannot cast
```

```
	// the memory as a struct so we have to walk it individually and shift
off
	// the bytes

	unsigned char *rt_data = ((u_int8_t*)rh) + sizeof(struct
ieee80211_radiotap_header);

	h->timestamp = time(NULL);

#ifdef LOGGING
	printf("Timestamp: %llu\n", h->timestamp);
#endif
	if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_TSFT)) {
		rt_data += sizeof(u_int64_t);
	}

	if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_FLAGS)) {
		rt_data += sizeof(u_int8_t);
	}

	if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_RATE)) {
		// rate is in 500 kbps
#ifdef LOGGING
		int rate = *((u_int8_t *)rt_data);
		printf("Radiotap data rate: %u Mb/s\n", TO_MBPS(rate));
#endif

		rt_data += sizeof(u_int8_t);
	}

	if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_CHANNEL)) {
		u_int16_t chan_freq = *((u_int16_t *)rt_data);
		rt_data += sizeof(u_int16_t);

		u_int16_t chan_flags = *((u_int16_t *)rt_data);
		rt_data += sizeof(u_int16_t);

#ifdef LOGGING
		printf("Radiotap channel: %u MHz, ", chan_freq);

		if(chan_flags & IEEE80211_CHAN_2GHZ) {
			printf("2 GHz band\n");
		}
		else if(chan_flags & IEEE80211_CHAN_5GHZ) {
			printf("5 GHz band\n");
		}

		if(chan_flags & IEEE80211_CHAN_PASSIVE) {
			printf("Radiotap channel: passive\n");
		}
#endif
	}

	if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_FHSS)) {
		//u_int8_t hop_set = *((u_int8_t *)rt_data);
		rt_data += sizeof(u_int8_t);

		//u_int8_t hop_pattern = *((u_int8_t *)rt_data);
		rt_data += sizeof(u_int8_t);
	}

	if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_DBM_ANTSIGNAL)) {
		h->rssi = *((int8_t *)rt_data);
```

```
                rt_data += sizeof(int8_t);

#ifdef LOGGING
                printf("Radiotap signal: %i dBm\n", h->rssi);
#endif
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_DBM_ANTNOISE)) {
                int8_t ant_noise = *((int8_t *)rt_data);
                rt_data += sizeof(int8_t);

#ifdef LOGGING
                printf("Radiotap noise: %i dBm\n", ant_noise);
#endif
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_LOCK_QUALITY)) {
                //u_int16_t lock = *((u_int16_t *)rt_data);
                rt_data += sizeof(u_int16_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_TX_ATTENUATION)) {
                //u_int16_t tx_atten = *((u_int16_t *)rt_data);
                rt_data += sizeof(u_int16_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_DB_TX_ATTENUATION)) {
                //u_int16_t tx_atten = *((u_int16_t *)rt_data);
                rt_data += sizeof(u_int16_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_DBM_TX_POWER)) {
                //int8_t tx_power = *((int8_t *)rt_data);
                rt_data += sizeof(int8_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_ANTENNA)) {
                //u_int8_t antenna = *((u_int8_t *)rt_data);
                rt_data += sizeof(u_int8_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_DB_ANTSIGNAL)) {
                //u_int8_t antenna = *((u_int8_t *)rt_data);
                rt_data += sizeof(u_int8_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_DB_ANTNOISE)) {
                //u_int8_t antenna = *((u_int8_t *)rt_data);
                rt_data += sizeof(u_int8_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_RX_FLAGS)) {
                //u_int16_t rx_flags = *((u_int16_t *)rt_data);
                rt_data += sizeof(u_int16_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_TX_FLAGS)) {
                //u_int16_t tx_flags = *((u_int16_t *)rt_data);
                rt_data += sizeof(u_int16_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_RTS_RETRIES)) {
                //u_int8_t rts = *((u_int8_t *)rt_data);
```

```
                rt_data += sizeof(u_int8_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_DATA_RETRIES)) {
                //u_int8_t retries = *((u_int8_t *)rt_data);
                rt_data += sizeof(u_int8_t);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_MCS)) {
                //u_int16_t mcs = *((u_int16_t *)rt_data);
                rt_data += (sizeof(u_int16_t) * 3);
        }

        if(BIT_SET(rh->it_present, IEEE80211_RADIOTAP_AMPDU_STATUS)) {
                //u_int64_t ampdu = *((u_int64_t *)rt_data);
                rt_data += (sizeof(u_int32_t) + sizeof(u_int16_t) +
sizeof(u_int8_t) + sizeof(u_int8_t));
       }

        // adding rh->it_len should get us to the very start of the 802.11 probe
request management data
        struct ieee80211_mgmt *wh = (struct ieee80211_mgmt *)((u_int8_t *)(packet
+ rh->it_len));

        for(int i = 0; i < ETH_ALEN; i++) {
                h->bssid[i] = wh->bssid[i];
                h->src[i] = wh->sa[i];
                h->dst[i] = wh->da[i];
        }

#ifdef LOGGING
        printf("SRC: %02x:%02x:%02x:%02x:%02x:%02x\n", wh->sa[0], wh->sa[1], wh-
>sa[2], wh->sa[3], wh->sa[4], wh->sa[5]);
        printf("DST: %02x:%02x:%02x:%02x:%02x:%02x\n", wh->da[0], wh->da[1], wh-
>da[2], wh->da[3], wh->da[4], wh->da[5]);
        printf("BSSID: %02x:%02x:%02x:%02x:%02x:%02x\n", wh->bssid[0], wh-
>bssid[1], wh->bssid[2], wh->bssid[3], wh->bssid[4], wh->bssid[5]);
#endif

        // what follows seq_ctrl are some variable length data items, and in this
particular case it is ssid and supported rates
        // we would only really be interested in the ssid to see that it is *not*
a base-station but rather a client broadcast

        // the format is u_int8_t tag, u_int8_t length then some u_int8_t data
for the length read in

        u_int8_t *tag = wh->u.probe_req.variable;
        tag++;

        int len = *tag;
        if(len > 0 && len <= MAX_SSID_LEN) {
        tag++;

#ifdef LOGGING
        printf("SSID: ");
#endif

        // copy over the length of the SSID
        if(len <= MAX_SSID_LEN) {
                strncpy(h->ssid, (const char *)tag, len);
                // add on the null byte
                h->ssid[len] = '\0';
```

```
        }
        else {
                strncpy(h->ssid, (const char *)tag, MAX_SSID_LEN);
                // add on the null byte
                h->ssid[MAX_SSID_LEN] = '\0';
        }

#ifdef LOGGING
        printf("%s\n", h->ssid);
#endif
        }

        pthread_mutex_lock(&lock);

        node *n = create();
        memcpy(n->h, h, sizeof(harvest));
        free(h);

        q->head = insert_back(n, q->head);
        q->count++;

        pthread_mutex_unlock(&lock);

#ifdef LOGGING
        packets_captured++;
        printf("Packets captured thus far: %llu, \nPacket queue count (insert):
%i\n", packets_captured, q->count);
#endif
    }
  }

  // this should never be reached
  if(db_path != NULL) {
    free(db_path);
  }

  if(prechosen_iface != NULL) {
    free(prechosen_iface);
  }

  while(q->head->next != NULL) {
    node *prev = q->head;
    q->head = q->head->next;
    free(prev);
  }

  return NULL;
}

int main(int argc, const char * argv[]) {
  pthread_t store_thread;
  pthread_t capture_thread;

        if(getuid() != UID_ROOT) {
                printf("You must be root to run this program.\n");
                exit(1);
        }

  struct ifaddrs *ifaces;
  struct ifaddrs *cur = NULL;
  if(getifaddrs(&ifaces) == 0) {
    cur = ifaces;
```

(3)

```
    while(cur->ifa_next != NULL) {
#ifdef AF_LINK
      if((strcmp(cur->ifa_name, EN0) == 0) && (cur->ifa_addr->sa_family ==
AF_LINK)) {
        const struct sockaddr_dl *dlAddr = (const struct sockaddr_dl *)cur-
>ifa_addr;
        const unsigned char *base = (const unsigned char *)&dlAddr-
>sdl_data[dlAddr->sdl_nlen];
        station_id = (u_int8_t)(base + (ETH_ALEN - 1));

        break;
      }
#endif

#ifdef AF_PACKET
      if((strcmp(cur->ifa_name, EN0) == 0) && (cur->ifa_addr->sa_family ==
AF_PACKET)) {
        struct sockaddr_ll *sl = (struct sockaddr_ll*)cur->ifa_addr;
        const unsigned char *base = (const unsigned char *)sl->sll_addr;
        station_id = (u_int8_t)base[(ETH_ALEN - 1)];

        break;
      }
#endif

      cur = cur->ifa_next;
    }

    freeifaddrs(ifaces);
  }
  else {
    station_id = UNKNOWN_STATION_ID;
  }

       // parse any arguments passed to harvestd
       for(int i = 0; i < argc; i++) {
              if(strcmp(argv[i], "-f") == 0) {
                     int len = strlen(argv[i + 1]);
      if(len < 1) {
        printf("You must enter a file path.\n");
        exit(1);
      }

                     db_path = (char *)malloc(sizeof(char) * len + 1); /* don't
forget the NULL byte */
                     strncpy(db_path, argv[i + 1], len);
      db_path[len] = '\0'; // add on the null byte

      printf("Overriding default database path: %s.\n", db_path);
              }
    else if(strcmp(argv[i], "-i") == 0) {
                     int len = strlen(argv[i + 1]);

      if(len < 1) {
        printf("You must enter a station ID.\n");
        exit(1);
      }
      else {
        u_int8_t id = atoi(argv[i + 1]);
        if(id < MAX_SIGNED_CHAR) {
          printf("Station ID must be between 0-255.\n");
          exit(1);
        }
```

```
        station_id = id;
      }
            }
    else if(strcmp(argv[i], "-n") == 0) {
      int len = strlen(argv[i + 1]);

      if(len < 1) {
        printf("You must enter an interface name.\n");
        exit(1);
      }

      prechosen_iface = (char *)malloc(sizeof(char) * len + 1); /* don't forget
the NULL byte */
                  strncpy(prechosen_iface, argv[i + 1], len);
      prechosen_iface[len] = '\0'; // add on the null byte

      printf("Using pre-chosen interface: %s.\n", prechosen_iface);
            }
     }

  printf("Using station ID: %i.\n", station_id);

  pthread_mutex_init(&lock, NULL);

  q = (queue *)malloc(sizeof(queue));
  q->head = NULL;
  q->head = NULL;
  q->count = 0;

  // init the capture thread and storage threads
  if(pthread_create(&capture_thread, NULL, capture_process_packets, NULL) != 0)
{
    pthread_mutex_destroy(&lock);
    printf("could not create capture thread");
    exit(1);
  }

  if(pthread_create(&store_thread, NULL, store_packets, NULL) != 0) {
    pthread_join(capture_thread, NULL);
    pthread_mutex_destroy(&lock);
    printf("could not create store thread");
    exit(1);
  }

  // wait forever for the capture thead
  pthread_join(capture_thread, NULL);
  pthread_join(store_thread, NULL);

  pthread_mutex_destroy(&lock);

  close_database(db_handle);
  free(q);

  return 0;
}

//
//  list.h
//  harvestd
//
//  Created by David R. Stites on 9/24/12.
//
```

```
//

#ifndef harvest_list_h
#define harvest_list_h

#include <string.h>
#include <stdlib.h>

#include "harvest.h"

typedef struct node {
  harvest *h;
  struct node *next;
} node;

typedef struct queue {
  node *head;
  unsigned int count;
} queue;

node *create();
node *insert_back(node *newNode, node *head);
node *remove_front(node *head);

#endif

//
//  list.c
//  harvestd
//
//  Created by David R. Stites on 9/24/12.
//
//

#include "list.h"

node *create() {
  node *n;

  n = (node *)malloc(sizeof(node));
  if(!n) {
    return NULL;
  }

  n->next = NULL;

  n->h = (harvest *)malloc(sizeof(harvest));
  memset(n->h, 0, sizeof(harvest));

       return n;
}

node *insert_back(node *newNode, node *head) {
  if(newNode == NULL) {
    return NULL;
  }

  if(head == NULL) {
    head = newNode;
    return head;
  }

  node *cur = head;
```

```
  while(cur->next != NULL) {
    cur = cur->next;
  }

  cur->next = newNode;

  return head;
}

node *remove_front(node *head) {
  if(head == NULL) {
    return NULL;
  }

  node *oldHead = head;
  head = head->next;

  free(oldHead->h);
  free(oldHead);

  return head;
}

//
//  harvest.h
//  harvestd
//
//  Created by David R. Stites on 9/27/12.
//
//

#ifndef harvest_harvest_h
#define harvest_harvest_h

#define MAX_SSID_LEN 32
#define SSID_BUF_SIZE 33

#pragma pack(1)
typedef struct harvest {
  u_int8_t msg_type; /* 1 byte */
  unsigned long long timestamp; /* 4 bytes */
  u_int8_t src[6]; /* 1 bytes x 6 = 6 bytes */
  u_int8_t dst[6]; /* 1 bytes x 6 = 6 bytes */
  u_int8_t bssid[6]; /* 1 bytes x 6 = 6 bytes */
  int8_t rssi; /* 1 bytes */
  char ssid[SSID_BUF_SIZE]; /* 33 bytes (one for NULL) */
  u_int8_t stn_id; /* 1 byte */
} harvest;
#pragma pack(0)

#endif

//
//  dstites_sqlite.h
//  harvestd
//
//  Created by David R. Stites on 9/26/12.
//
//

#ifndef harvest_dstites_sqlite_h
#define harvest_dstites_sqlite_h
```

```
#define CALL_SQLITE(f)                                                \
{                                                                       \
  int i;                                                              \
  i = sqlite3_ ## f;                                                  \
  if (i != SQLITE_OK) {                                               \
    fprintf (stderr, "%s failed with status %d: %s\n",  \
      #f, i, sqlite3_errmsg (db_handle));                             \
    exit (1);                                                         \
  }                                                                     \
}                                                                       \

#define CALL_SQLITE_EXPECT(f,x)                                       \
{                                                                       \
  int i;                                                              \
  i = sqlite3_ ## f;                                                  \
  if (i != SQLITE_ ## x) {                                            \
    fprintf (stderr, "%s failed with status %d: %s\n",  \
      #f, i, sqlite3_errmsg (db_handle));                             \
    exit (1);                                                         \
  }                                                                     \
}

#endif
```

APPENDIX B: DISCOVERED SSIDS (PARTIAL LISTING)

AppleWiFi	Ross	aalloo-guest	mobilepoint
Misty Mountain (5 GHz)	TC_WL	aalloo 5GHz	livedoor-web
Ming's iMac	PonPaNet	Jun	33
Bearded Mail	Academy-Guest	MICHAEL-PC_Network	Maya
AppleWiFiSecure	HOME-8B28	Tehya Misr	CODY8898
Misty Mountain	JST	A88551D4GL08ALWLan	webOS Network BF:32:81
attwifi	JustFine	PLP	2WIRE588
Adorn (5 GHz)	Genius Room	Home	Nocimed
Apple Demo	yippy	6X8XL	Shop1
Air-	SFO-Public	KASEM&MOHAMED	serengeti
haahoosal	Pizza	sjcfreewifi	lab2
Apple Store	BestBuy	LAX-WiFi	Air-1BFDEB
Houwen's iMac	sbmc	R28	Air-31C838
Zontar	HomeNetFast	Wael-Zone	WOS
jbl-cisco	iPad	HOME-DBD2	JuniperWirelessNetwork
greentea w/redbull 5GHz		crowne_plaza_irvine_38	Doghouse
dpak	crib	2WIRE522	Jean Weingarten

Adorn	Abby2	2WIRE008	WLESS
JBWifi	2WIRE304	ANDROMEDA	RamadaMaingate
Zontar 5GHz	deathstar	abbylan	Stafford's Network
CommuteWiFi	WIFI-AIRPORT	Renaissance Wireless Guest	Metro_WiFi
330High	CommuteWiFi-a	RTC Public	District_Guest
Danny Iacono's iMac	commute_evaluation	aloft_Plano	Courtyard_Conf
EM50	VistaCast	Guest	SilverOak
Transplant	BJ's Guest WiFi	Dream-Public	Verizon SCH-LC11 05c3 Secure
Home Depot	Nicie	Pier57	Nick
The Truth	BELKIN54G	JCP	WONDERS
greentea	Ritz-Carlton Wireless	CWBH Public	Casa Tua
alsoiloveyou	daisy	ICU95129ICU	thu5att
Rooster	dlink	ICU95129icu	covidien-guest
BeardedMail		Kitty	BikramYoga
surfview	Hi-Fi Extreme	Casa Blanca	Air-C16CB5
Pearland	San.Diego.Airport.Free.WIFI	matthew	Alpha
CoreTech	SFAPT	westfield	cbcmilpitas
Sonny Hoàng’s iPhone	BJWnet	ATT896	Air-969011
Tahoe House	36 Hartford Guest	COG-Hotspot2	HomeCon

CoreTuna	Chamber7-WiFi	Purple Pony	ixoras-ap1
brandon-N	WireSharc	LUNAR	horizon2
workhorse	Ubiquity	FamBam	247-LDAP
Pele	VastConcept	Verizon DROIDX 5909	dobpcj
SHGuestNet	Sharks Ice	MCREW6	Gulfstream Wireless
Sweethome	LBL3	PurpleCheetah-guest	VP-Guest
NETGEAR07	forGames	maldonado1969-guest	Hyatt_LostPines
elmosys	2WIRE852	MOT-1-B2	2012 ASC GSM
OpenDearborn	SFO-WiFi	??	fourpoints
TL-WR841Nv5.2	Sawkins	HOME-DF58	Parker Meridien
310-BIGHORN	Plern's Network	boddu	Phoenician
KenwoodOaksGuestHouse_ure	GlobalSuiteWireless	smc-guest	Phoenician Conference
WWDC2012	RadissonRochesterRiverside	Nacho	WESTIN-GUEST
flypdx	vish	agc	HyattGuestroom
Kasia	Innpublic	kimpton	WWLS
G12network-guest	MasterLeague	The Taylor Airport	Aircell Broadband
Elmosys	GTwpa	AA	MLS2012
charliebrown	A BAR FREE WIFI	Orpheum	EnglishInn21
<Fighting Irish>	chadwick-guest	Guest - JW Marriott	SRIVAIKUNTAM
wireless	NETGEARKKD	Biltmore-Wireless	linksys-g
VastConceptz	linkmax	Conference - JW Marriott	ResidenceInn-Guest
gogoinflight	Gorringe	OCI	Merit-guest

2WIRE032	92-304-uc	Renaissance Wireless Meeting	PING1541
Caltech BeaverNet	White-309	ctevents_redsky	SSSS
Regal Beagle	Goku's Network	CBC	VAIKUNTAM
Zeratul's Legacy	fantismo	_PRAY_PSALM_51	AG_Guest
JennyPenny	houwen_n92_1	Jesus Saves-Repent and Believe	NETGEAR_WIRELESS
ljjiang’s iPad	AndroidLibertyAP	MotionMedical	Sahana
ARENAINN	Outer Rim	CupertinoInn1	Samsandy
Roulette	Comfort Inn Marina	EBCWiFi	Sprint MiFi4082 23F
Billy Shears	AUX Wireless Network	TimeOut	Sprint MiFi4082 21E
Bania	miramar2	Hostal Grau	Telenav-phone access
linksys	Charlotte	ChicoAirport	Ballys-Rooms-Cox
SCFAM2012	ChinaNet	Pi	TelenavSoftwareAP
05Z404734362	Tempest	AirHead	TelenavSoftware-Meeting
Gyatt NET	Beaker	airhead	g1603
Gyatt NETg	londonlan	SAINT JACQUES	ASUS
pure lounge	SappBrosTS	ibahn	Trybom
4dcow	Lattanzio	Crowne Plaza Public Wireless	2WIRECHEESY
farkU	Valery's Wi-Fi Network	Virus Detected	Armadillo Willys

banlieue	coffeesociety	Largo	TREEHOUSE
wabe	Travelodge Anaheim	SUM_SYN_OPEN	ToyotaCustomerWiFi
jNet	ARG_Test	swisscom	garfield-guest
sleepyhouse	Joelga	Ahwahnee Hotel	The Lounge
SMG-guest	Heroes	Michael Wilhelm's iPhone	minigourmet2
Kris and Niu Network	bawb5225 Network	WG phone	ArmadilloWillys
BerlitzN-guest	AirForce	WestinGuestRooms	4416 0450
Alpine Meadows AP1	SpaceInvader	CUSD_Guest	MARYBETH-PC
FreeSMTP	UMTS Lab	flysacramento	cva
beret	J2-carry-315	Internet	WIFI
Bania (5 GHz)	sdlab	pid-guest	ThunderCougarFalcon Bird
Fyzzle (5GHz)	plumtree	Nassau Airport Free WiFi	FunNetwork
1spiderpigtrois	5G@Greensprings-112H	H Porta Fira	ECBS-guest
scc_public	UChi's lion	VMworld2012	McCarran WiFi
RRM-Guest	Delfina-GuestRoom-WiFi	VMworld_5Ghz	sca
Apple WIFI	hhonors	Gotterdammerung	HYATT
JoshNet	uchi-wireless-5G	Best Buy Wireless	XtremeLabs
DataValet	MattPeter	SH4BQ	rugeriver1
lucky77	shs-guest	Gotterdamerung	Stanford
uNet-5GHz	HappyHome	Mandalay	Xtreme Labs Palo Alto

WonderDogWireless	Wireless	2012 ATA MCE	Motorola
MandarinOriental	cestmonwifiamoi	Wotan	Xtreme Labs Guest Network
J2	EventStaff	OCI Guest	Pearl Harbor
tmobile	hamptons6102	staff	AWG-WiFi
Shaffer	PandaExpress	2WIRE618	@yvrairport
demo	ritkumar	Blue Coat	Hdb14Net
N94_PVT_Enrique_179	JKnetwork	Marriott-Conference	Lot104
52 Stations	2WIRE968	Marriott-GUESTROOM	Galaxy
2WIRE155	Verizon SCH-LC11 f14c Secure	Kimpton	Wormhole
Cujo391	SURFBOARD	jcp	Domain Hotel
Best Western	SpeedStream	GIOVation	Felix-PC-Wireless
mansharkman (5 GHz)	Linkpath2.4	JWMarriott_GUEST	MarinaDelRey
2WIRE787	Choose Happy	Oracle_wifi	Ramada
Rii	Andromeda	PSAV_Event_Solutions	Sprint MiFi4082 21E Secure
LBI Shore	andromeda	production	Dont Steal My Internets
united_club	AMAT_Prod	George	dlink3333
Cisco52703	XiaoBai	NISSAN	GSO_MC
ELITE	ANAG	thc-corp	2WIRE137
42gNet	Serene	VTA	Macysfreewifi
rei-guest	Georgian Court Meetings	horizon1	Boingo Hotspot
MMI-Internet	Slingbox Demo	FISHFREE	GoogleGuest
Apple Network 0026a1	pga	brettonwoods	Washington Dulles WiFi

wehen	Sammy	Max's	LMM_Airport_Open_ Hotspot
JS Fields	basewireless_Upper Village186-03	ethostream42	Hotel_El_Convento
Sunny_Guest	nvwrls2	VersatilePower	DosAngelesdelMar
a series of tubes	NETGEAR	Corp2	omega
Apple Airport Network	yodel	Coffee Fellows Tal	PPCA24
Traffic Jam	nvwrls	AOIC	belkin54g
glfreak	CoolSpot	Annis Lake 17	Blue Line Public WiFi
piinknet	2WIRE178	PPTC2	SpeedStream_EXT
Living Room	@Home	2WIRE864	#LG@Vo1P*Service&
2WIRE668	TechShop_San_Jose _WiFi	MAIN	Artyom
sanddollar789	telenav	zeis	Saggar
torretes	aalloo	Relievant	HOME-F072
Culibri	BM-Wireless	PPTC3	
WakaWaka	UCLA_SECURE_R ES	ATT245	