NTNU

Norwegian University of
Science and Technology

Detecting MAC Spoofing Attacks in 802.11 Networks through Fingerprinting on the MAC Layer

Christer Idland

Master of Science in Communication Technology
Submission date: June 2011
Supervisor: Stig Frode Mjølsnes, ITEM
Co-supervisor: Martin Eian, ITEM
Thomas Jelle, Trådløse Trondheim

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Name of Student Christer Idland

MAC spoofing attacks are a significant threat to open 802.11 networks because they constitute an identity theft and are easy to perform as shown in [1]. Wireless Trondheim, who presented the general problem of MAC spoofing attacks that lead to this thesis have an urban network with very mobile clients. Their network infrastructure, usage patterns and the surrounding environment make it difficult to use physical layer parameters to detect MAC spoofing attacks as concluded in [2].

Idland was able to improve an already existing intrusion detection system (IDS) based on logical tests in the MAC layer [1]. The IDS developed showed good results, particularly in detecting attacks of the freeloader type — that is when both attacker and victim are connected simultaneously. On the other hand, no good solution exists to detect attacks where the legitimate client is no longer online. The two main scenarios is the session hijacking attack, where the attacker forces the victim offline, and the wait-for-availability attack where the attacker waits until the legitimate client leaves the network.

The heterogeneity of implementations in the 802.11 MAC protocol is demonstrated in [3] and [4]. The problem is then to determine whether different implementations of the 802.11 protocol can provide means to fingerprint devices on the MAC layer. And more importantly if these fingerprints can be used to distinguish between an attacker and a victim in the above mentioned scenarios in an efficient way and with high probability.

Assignment Given 17.01.2010

Supervisors Professor Stig F. Mjølsnes, Martin Eian and Thomas Jelle

Detecting MAC Spoofing Attacks in 802.11 Networks through Fingerprinting on the MAC Layer

TTM4905 — Master's Thesis

by

Christer Idland
christer.idland@gmail.com

June 13, 2011

Supervisors: Professor Stig-Frode Mjølsnes, Martin Eian and Thomas Jelle

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

Abstract

In order to provide hassle-free connection options many wireless local area network (WLAN) providers choose to have their networks completely open. In other words there is no password required in order to connect. Such open configurations do not provide any security features on the wireless medium, but are often implemented with other solutions as captive portals. A captive portal forces a Hypertext Transfer Protocol (HTTP) client to see a certain webpage, usually for authentication purposes. All other packets are blocked. Once authenticated, the client's medium access control (MAC) address is whitelisted and he will have access to the Internet.

The MAC spoofing attack is easy to perform in open networks, see Appendix A. This attack can have severe consequences as the attacker masquerades as a legitimate client, potentially getting the victim caught for crime done by the attacker. The preferred way to handle these attacks has been through detection, as it can be done on the server side without complicating anything for the user. Effective and reliable detection techniques for plain and quality of service enabled 802.11 networks exists [1, 5]. However, no good solution exists to detect attacks when the legitimate client is no longer connected. The two main scenarios are the session hijacking attack, where the attacker forces the victim offline, and the wait-for-availability attack where the attacker waits until the legitimate client leaves the network.

An algorithm based on MAC layer fingerprinting was developed to detect the class of attacks where attacker and victim are not connected simultaneously. A fingerprint is based on the behavior of a station (STA), and each STA's behavior varies due to implementation differences of the 802.11 protocol. Experiments in a real network was performed with 11 different STAs in order to determine the fingerprints. The results show that on average 2.82 of the 8 fingerprinting properties were different when comparing two fingerprints.

The fingerprinting algorithm developed is capable of passively creating a fingerprint of wireless STAs without specialized equipment in realistic network conditions. Fingerprints from different STAs are unique with high probability, even when there are little data available. In addition, the technique used is accurate, fast, and requires no pre-computed databases. The algorithm used in combination with the intrusion detection system developed by Idland [1] is now able to detect all of the five different MAC spoofing attacks described in Section 2.6.2.

Preface

This report is the final product of the research I have been doing in my master's thesis, TTM4905, during the spring of 2011. The master's thesis is the final work required as a part of my five year program studying for a Master of Science in Communication Technology with specialization in information security. The work has been carried out at the Department of Telematics at the Norwegian University of Science and Technology (NTNU), and in my home town Sandnes where I spent most of the time this semester.

The assignment was proposed by Wireless Trondheim, and the thesis has been a continuation of the work done by myself in my specialization project at NTNU in 2010 [1].

I would like to thank the staff at Wireless Trondheim for lending me equipment to perform some of the experiments in Sandnes. In addition I will thank them for their guidance with the practical work, as well as helping me through e-mail when needed. Your technical expertise has been most appreciated. Special thanks is directed to my supervisor Martin Eian who have contributed with invaluable input throughout the thesis. Your feedback and interest in my work have made a big difference, and it is much appreciated.

June 13, 2011

Contents

List of Algorithms

List of Figures

List of Tables

Acronyms and Initialisms

ACK	acknowledgment
AID	association ID
AM	active mode
AP	access point
CCX	Cisco Compatible Extensions
CKIP	Cisco Key Integrity Protocol
CRC	cyclic redundancy check
CTS	clear to send
DA	destination address
DF	don't fragment
DoS	denial of service
DS	distribution system
DSCP	Differentiated Services Code Point
ERP	extended rate PHY
FCS	frame check sequence
FPA	fingerprinting algorithm
FPP	fingerprinting program
GHz	gigahertz
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure

ID	identifier
IDS	intrusion detection system
IE	Windows Internet Explorer
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPS	intrusion prevention system
Kbps	kilobit per second
LAN	local area network
LAP	lightweight access point
LCN	Local Computer Networks
LLC	logical link control
MAC	medium access control
MITM	man in the middle
MPDU	MAC protocol data unit
MSB	most significant bit
MSDU	MAC service data unit
NIC	network interface controller
NTNU	Norwegian University of Science and Technology
OS	operating system
OSI	Open Systems Interconnection
P/W	PROSet/Wireless
PDA	personal digital assistant
PDF	Portable Document Format
PHY	physical layer
PLCP	physical layer convergence procedure

PMD	physical medium dependent
PS	power save
QoS	quality of service
RA	receiving STA address
RAID	Recent Advances in Intrusion Detection
RSN	Robust Security Network
RSS	received signal strengt
RTS/CTS	request to send / clear to send
RTS	request to send
RTT	round trip time
RX	receive or receiver
SA	source address
SQN	sequence number
SSID	service set identifier
SSL	Secure Sockets Layer
STA	station
TA	transmitting STA address
TCP/IP	Internet Protocol Suite
TCP	Transmission Control Protocol
TID	traffic identifier
TIM	traffic indication map
TSF	timing synchronization function
TSFT	TSF timer
TTL	time to live
TX	transmit or transmitter

UDP User Datagram Protocol

URL Uniform Resource Locator

VPN virtual private network

VTC Vehicular Technology Conference

WLAN wireless local area network

WLC wireless LAN controller

WME Wireless Multimedia Extensions, also known as WMM

WMM Wi-Fi Multimedia

WrT Wireless Trondheim

Chapter 1

Introduction

This chapter will provide the background of the problems described in the abstract as well as give a general introduction to the relevant topics. It will start by explaining why open 802.11 networks are commonly deployed before the need for intrusion detection systems (IDSs) in such networks are presented. This will be followed by an introduction to Wireless Trondheim and a thorough description of the problem at hand. Then, an overview of relevant work done by others will be given, and finally the chapter will end with an outline of the rest of the report.

1.1 Motivation

1.1.1 The Popularity of 802.11 Networks

A wireless network is a type of computer network connecting devices without any kind of wiring or cables. The two most common types of wireless networks would be the ones used for mobile telecommunications (cellular phones) and wireless local area networks (WLANs). WLANs are based on the 802.11 standard and are often referred to as Wi-Fi networks[1] or hot-spots, and this is the type of network that will be considered in this thesis. Although a network in itself could have some value, these networks usually provide access to the Internet as their main service. Most mobile devices produced over the last five years such as smartphones, laptops, personal digital assistants (PDAs), and tablets have the ability to connect to Wi-Fi networks.

WLANs are found almost everywhere where people move, this includes stores, offices, cafés, schools, private homes and sometimes even downtown outdoors areas as the one found in Trondheim, Norway. People want to be connected to the Internet wherever they are and as a result hot-spots are commonplace in public areas. These

[1] Wi-Fi networks are 802.11 networks that are based on components certified by the Wi-Fi Alliance. Wi-Fi is commonly, but incorrectly used as a synonym for 802.11.

public hot-spots can be separated into two types based on the payment model.

Access to the network can be free of charge to consumers as a complementary service, or access to the network can be sold as a service of its own. The first approach is commonly found in coffee shops, and a good example is Starbucks which is known for their free Wi-Fi access. The payment model on the other hand is typically found in most airports in North America and Europe. Common for both approaches is the need to be convenient and easy. Users expect to be able to connect with any kind of Wi-Fi equipment without the hassle of a complex configuration or other tedious information entering.

In order to provide hassle-free connection options to their network many WLAN providers choose to have their networks completely open. This implies that there is no encryption, and no password required to connect to the network. Such open configurations do not provide any security features on the wireless medium, but are often implemented with other solutions as captive portals. A captive portal forces an Hypertext Transfer Protocol (HTTP) client to see a special web page (usually for authentication purposes) before gaining access to the Internet. All other packets are blocked. Once the client enters proper credentials, the captive portal will mark the clients medium access control (MAC) address on a whitelist. After this subsequent traffic will no longer be blocked, and the associated client will be able to access the Internet. The login credentials are often temporary and acquired through a third-party as a complementary service (hotel, coffee shop) or by paying a fee. Using your credentials from other systems is sometimes possible, and this feature is present in Wireless Trondheim (WrT) where students can use their unversity credentials to log-in.

1.1.2 Security Implications with Open 802.11 Networks

As there are no encryption in open 802.11 networks everything a station (STA) sends will be available for everyone in the vicinity, this clearly presents a confidentiality issue. Authentication is either non-existing or barely existing as in the case of captive portals. The confidentiality issue is not as bad as it first looks, this is due to the fact that most sensitive web-pages such as e-mail, bank, web-shops etc. will be using HTTP Secure (HTTPS) and thereby encryption on the application layer. Confidentiality can also be acquired on other services by using a virtual private network (VPN) solution.

The problem with the authentication is identification, or the ability to link an action performed on the network to a STA, its user, and in the end to a specific person. The ability to successfully do this becomes especially important in criminal cases. One wants to make sure that if an illegal act has been carried out from a

client's credentials it was in fact the legitimate client who performed the act, and not someone else masquerading as him. Stealing another user's credentials is known as an identity attack.

The latter is the rationale for the use of IDSs as a mean to avoid accusing the wrong person for a crime he did not commit. More details of how IDSs can be used in this case will follow.

1.1.3 Wireless Trondheim

WrT began as a project for research and development at the NTNU in 2005. In the autumn of 2006, several public and private partners came together to set the project in motion and Trådløse Trondheim AS (Wireless Trondheim) was founded [7].

The main objectives of WrT are to build and deliver easy wireless Internet access in the city of Trondheim, Norway. In addition they are doing research and development of new wireless and mobile services. WrT hosts a quality of service (QoS) enabled 802.11 network that covers most of the downtown area in Trondheim, both outdoors and indoors. This is accomplished by positioning lightweight access points (LAPs) in strategic locations, these LAPs are controlled by wireless LAN controllers (WLCs). A figure depicting this setup is presented in Section 3.1.1.

Access to the Internet is gained after a successful login through their captive portal. Students can login with their university credentials to get free access, and anyone can pay a small fee to get access for a specific time frame. It is important for WrT that it is possible to connect to their network with any Wi-Fi equipment, this includes laptops, smartphones, PDAs, tablets and so on. This puts a restriction on what is possible to do on the client side in terms of preventing attacks. For instance, the active Secure Sockets Layer (SSL) window technique [8] cannot be deployed as many smartphones cannot run two browser windows at the same time.

1.2 The Problem

Identity attacks in open 802.11 networks with a captive portal are easy to perform, even with limited knowledge and equipment. One such attack is called the wait-for-availability attack, for details on how to perform it see Appendix A.2. As indicated earlier identity attacks can have serious consequences when a malicious user performs some illegal activity, as for instance browsing of child sexual abuse images, while using the identity of a legitimate client.

WrT is concerned about this type of identity attacks and they are the ones who initially presented this problem as a potential task for a master's thesis. As a result I will focus on the specific network type that is deployed in WrT, as well as take

into consideration the specific restriction due to their aim to be a easy network to connect to for all kinds of devices. There exists ways to prevent these kinds of attacks, and some details will be given on this topic in the next chapter. The main problem with these prevention techniques is that they require the user to be active (setting up encryption keys or certificates) or they require that the STA is relatively advanced (i.e. able to run two browser windows at once). None of these options are desirable when you want to sell quick and easy Internet access to a broad range of devices. That is why detection has been, and will be in this project, the preferred way to deal with this problem. Detection can be done on the provider side of the network without complicating anything for the users.

Plain MAC sequence number (SQN) analysis has previously been proven to be an effective and reliable detection technique in 802.11 networks [5]. More sophisticated detection techniques that also works in Wi-Fi Multimedia (WMM) enabled networks have been specifically designed to meet the demands of WrT [1]. The IDS developed by Idland showed good results, particularly in detecting attacks of the freeloader type — that is when both attacker and victim are connected and generate traffic simultaneously. On the other hand, no good solution exists to detect attacks where the legitimate client is no longer connected. The two main scenarios are the session hijacking attack, where the attacker forces the victim offline, and the wait-for-availability attack where the attacker waits until the legitimate client leaves the network. More details will be presented in Section 2.6.

Some interesting papers on MAC layer fingerprinting and homogenity on the MAC layer in 802.11 have been published [3, 4, 9]. Fingerprinting is the act of collecting externally observable characteristics from a specific source in order to identify it. Based on the information in the above mentioned papers it is possible that a detection based on fingerprinting could in fact prove viable against the types of attacks in question.

The problem can be stated as:

How to detect a session hijacking attack or a wait-for-availability attack on an open 802.11 network.

In collaboration with WrT and my supervisors from NTNU we formulated the following task description.

Determine whether different implementations of the 802.11 protocol can provide means to fingerprint devices on the MAC layer. And more importantly if these fingerprints can be used to distinguish between an attacker and a victim in the above mentioned scenarios in an efficient way and with high probability.

The task description describes a possible solution to the problem with the use of

fingerprinting and thereby narrows the scope as there are certainly several ways of addressing this problem. The scope is further narrowed to not include work aimed at spoofing a fingerprint or altering a STA's behavior in order to match a specific fingerprint. Even though that would have been relevant work, this prioritization is necessary due to the limited time available for this thesis. The main priority is therefore to develop an algorithm to fingerprint devices on the MAC layer, and thoroughly test this algorithm with real STAs in order to address the issues pointed out in the task description above.

While the problem and task description are both focused on WrT, the work done in this thesis is general enough to be relevant for the vast majority of open 802.11 networks. The fact that many 802.11 networks have a strong focus on ease of connectivity and hassle-free setup on different kinds of equipment, makes the work done here relevant in a general sense.

1.3 Method

The work done in this thesis was completed in four different phases. Acquire information, developing an algorithm, performing the experiments and analyzing & presenting. The details of these phases and the rationale behind will soon be explained. These phases were performed in the order explained below, but they were revisited at least once as the work was done in an iterative manner. In the first iteration new information was discovered while analyzing the results from the experiments, this information was in turn included in the algorithm which again required new tests, analyzing and presentation.

1.3.1 Phase One — Acquire Information

The first phase consisted of acquiring information and knowledge about the relevant special field. More precisely an understanding of the current state of the art in fingerprinting, MAC spoofing detection and 802.11 implementation differences were obtained.

1.3.2 Phase Two — Developing an Algorithm

The second phase consisted of constructing an algorithm for fingerprinting on the MAC layer. That included determining which properties in a STA's behavior and in the 802.11 protocol that could be used for fingerprinting, creating individual tests or small algorithms for each such property, and implementing the most time consuming of these tests in Perl for easy execution when dealing with large amounts of data.

The work done in phase two naturally depended on the information obtained in phase one.

1.3.3 Phase Three — Performing the Experiments

As the potential solution itself relies on implementation differences of the 802.11 protocol it became clear that the method used should be based on empirical data and not solely theoretical information or simulation. As a result, experiments in a real network (WrT) with real STAs was performed in the third phase in order to be able to evaluate the fingerprinting algorithm.

1.3.4 Phase Four — Analyzing & Presenting

Phase four, the last phase, started with analyzing the data obtained through the experiments in the previous phase. Then, the majority of the work was done in writing this report, making sure that the work done in this and all the previous phases was presented in a logical and readable way.

1.4 Contributions

The main contributions of this thesis are the design, implementation, and evaluation of a passive wireless fingerprinting technique to enable detection of certain types of MAC spoofing attacks.

The fingerprinting technique is based on ideas found in [3] and [4] and the discussion of what to include and why can be found in Section 3.4. A thorough walkthrough of the algorithm developed including pseudocode can be seen in Section 3.5. The experiments and the rationale behind them will be presented in Chapter 4. The results and conclusion from the experiments will be presented in the result chapter and conclusion chapter respectively. The source code, in its entirety, can be seen in Appendix C.1 (`FPP.pl`).

1.5 Related Work

The work covered in this report is a continuation of the work done by Idland in his specialization project *Detecting Identity Thefts in QoS Enabled Open 802.11 Wireless Networks* [1] and it is necessarily very relevant related work. In addition to that one report this section will present other work done on relevant attacks, ways to prevent them, different detection techniques as well as work done on fingerprinting on the MAC layer.

1.5.1 Attacks on Wireless Networks

As wireless networks are very popular there exist numerous attacks on the confidentiality and authenticity of the network. The most relevant attacks for this thesis are found in a class called MAC spoofing attacks. As the name implies these attacks is based on spoofing another STA's MAC address as one's own. There are three main versions of this attack, the MAC freeloader attack, the session hijacking attack and the wait-for-availability attack and they will all be explained in greater detail in the next chapter.

The necessary technical information and descriptions of how to execute the above mentioned attacks are easily found. The attacks performed in the experiments in this thesis were based on the general information about 802.11 found in Edney and Arbaugh's book, *Real 802.11 Security* [10], combined with an overall description of the different types of MAC spoofing attacks found in [8].

1.5.2 Preventing Attacks through Active Methods

A promising solution called active SSL window is presented in [8]. This could effectively prevent both the session hijacking attack and the freeloader attack. The downside is that this technique requires that the STA always run a browser window with the SSL connection. As a result a STA must run two windows at the same time if the user want to browse the web, a feature that is currently not supported by a host of smartphones and other Wi-Fi equipment (tested by employees at WrT).

Despite this technique's effectiveness against the exact attacks this thesis addresses it is not very relevant as it is restricted to STAs as laptops and possibly more advanced tablets. This is due to WrT's and other WLAN operators goal to not limit the type of equipment that can connect to their network.

1.5.3 Detection Based on Physical Properties

Two detection techniques based on physical parameters were proposed in [5],they were based on monitoring received signal strengt (RSS) and monitoring round trip time (RTT) of the request to send / clear to send (RTS/CTS) handshake. The accuracy of these techniques were published a year later [11]. These techniques were aimed towards detecting MAC spoofing attacks and the results are promising with one important constraint. Their experiments were performed in static environments, including an office premises, and are therefore not necessarily applicable to our setting in WrT.

In 2009 Pedersen performed experiments with RSS techniques in urban, dynamic and mobile environments as the one found in WrT in his master's thesis [2]. The

conclusion from his work is that RSS is not at all reliable and conclusive when moving away from office landscapes and lab environments, and into urban environments. He goes on to say:

> The rate of failure stated here may be adequate concerning location based services, but an IDS should depend on reliable and conclusive parameters in order to trigger alarms on attacks [2].

Therefore, detection based on physical parameters will not be further pursued in this thesis.

1.5.4 Detection Based on Logical Properties

The paper *Sequence Number-Based MAC Address Spoof Detection* [12] explains the basics behind the SQN detection method used to counter the MAC freeloader attack. This paper is also the foundation for a IDS developed by Holgernes [13] and further improved by Idland [1]. Holgernes and Idland augments the basic detection based on MAC sequence numbers with state machine analysis and QoS validation respectively.

The IDS is tested in WrT's network with good results, but it is only applicable to scenarios when the attacker and victim are connected simultaneously. Nevertheless it is very relevant work and this work can be seen as a continuation of that work expanding the detection to other scenarios.

1.5.5 Heterogeneity in 802.11 Devices and Implementations

It turns out there are quite some heterogeneity in 802.11 implementations and the behavior of different devices. These differences are in fact what makes fingerprinting on the MAC layer possible. This subsection will give a short overview of the most relevant work done in this area and explain how it has affected and motivated the fingerprinting algorithm developed in this thesis.

Wireless Driver Fingerprinting In the paper by Franklin et al. [9] the authors exploit the fact that the algorithm used for scanning channels for access points (APs) is not explicitly defined in the 802.11 protocol. As it is not explicitly defined different manufacturers have developed different algorithms. The authors of the paper developed a method based on statistical analysis of the inter-frame timing of transmitted probe requests in order to identify a specific driver.

They call this *Wireless Device Driver Fingerprinting* and concludes that the majority of wireless drivers do have a distinct fingerprint. The focus in this paper is on the attacker's perspective, that it is easy for an attacker to determine which

driver a given STA is using, and then be able to perform driver specific attacks against that STA.

Heterogeneity in the 802.11 Protocol Gopinath et al. gives an empirical analysis of the heterogeneity in the 802.11 protocol in their paper [3]. The differences they identify are; different random backoff algorithms, whether or not the duration field is honored, the calculation of the duration field, reassociation latency, rate switching behavior and vendor extensions in the association request. They show that these differences can result in unfair bandwidth allocation and poor network utilization as the STAs do not all act the same.

More interesting for this thesis is the fact that they suggest that these differences can be used in device fingerprinting in order to detect for instance a MAC spoofing attack. They do not however explore that possibility further. Some of the differences in the 802.11 protocol identified in this paper will be part of the algorithm developed, while others not.

Null Data Behavior The paper *On Security Vulnerabilities of Null Data Frames in IEEE 802.11 based WLANs* written by Gu et al. [4] is very important to the work done in this thesis. The authors create seven rules to identify different behavior regarding the Null Data frames. These rules form the basis of the algorithm developed in this thesis. The authors of the paper focus on the attacker's perspective and the fact that this fingerprinting allows an attacker to recognize a user, and to determine his location at a given time. Location is in this case limited to a WLAN, for instance at a café, at home or at school. They call it *Implementation based Fingerprinting Attack*. In this thesis the rules will be used the other way around, namely in order to detect attacks and not aid them.

1.6 Report Outline

The outline for the remaining of the report is presented below.

Chapter 2 provides the relevant background theory to the work done in this thesis. This includes detailed information about the 802.11 protocol, an overview of relevant attacks and methods to prevent such attacks. An explanation of the rationale behind IDSs is given as well as an explanation of what fingerprinting is and how it can be done on the MAC layer. If you have a background in wireless security you might want to skim or even skip the first part of this chapter, the fingerprinting part might still prove useful.

Chapter 3 presents the background for why fingerprinting can be used as a viable detection method and a detailed description of how the fingerprinting algorithm is

developed. The last part of this chapter includes pseudocode and descriptions of the algorithm and makes up a major part of the contributions in this thesis.

Chapter 4 elaborates on how the experiments was performed, this includes the physical setup, an overview of the different STAs used in the experiments and the test scenarios. Then, *Chapter 5* presents the results obtained through performing the scenarios as described in the previous chapter.

This is followed by *Chapter 6* which turns to the discussion and analysis of the results obtained, their uniqueness and validity, as well as how the fingerprinting algorithm would do as a detection method in a real world IDS.

Chapter 7 outlines the potential for further research within the same field of study. Finally, *Chapter 8* concludes the work done in this thesis and relates it to the problem described in Section 1.2.

Chapter 2

Theoretical Background

This chapter will provide the necessary theoretical background in order to understand the relevant attacks and how fingerprinting on the MAC layers is possible due to the nature of the 802.11 protocol. The chapter will start with a general overview of the 802.11 standard before a more detailed review will be given on some of the more relevant frames used in the protocol. Finally the chapter will present the most relevant attacks on open 802.11 networks and the idea behind IDSs.

2.1 General Overview of IEEE 802.11

2.1.1 The 802.11 Set of Standards and Wi-Fi

The Institute of Electrical and Electronics Engineers (IEEE) 802.11 is a set of standards for wireless networking created and maintained by the IEEE. The most well known protocols are the 802.11b and 802.11g, which are amendments to the original standard. The base current version is the IEEE 802.11-2007 (referred to as 802.11 in this report) and it includes the above mentioned amendments in addition to others including 802.11e (QoS) and 802.11i (security).

802.11 operates in the data link layer and the physical layer (PHY) of the Open Systems Interconnection (OSI) model provided in Figure 2.1. In 802.11 the data link layer is a composite layer, consisting of the sublayers logical link control (LLC) and MAC. As evident from Figure 2.1 the MAC layer is the layer above the PHY in the OSI model. On the physical layer 802.11b and 802.11g use the 2.4 gigahertz (GHz) band. The 802.11a use the 5 GHz band while the newer amendment 802.11n operates in both the 2.4 GHz and 5 GHz bands.

Complying with the standard should ensure that your equipment can communicate correctly with other equipment also adhering to the standard, and together form a wireless network. Nevertheless there exist independent certifications for interoperability, the most popular is Wi-Fi. The Wi-Fi Alliance is a non-profit organization

and their certification includes rigorous testing to ensure that different devices can interoperate in a wide variety of configurations. Approved equipment can then be branded with the Wi-Fi logo to promote their interoperability. The terms Wi-Fi and 802.11 are therefore sometimes, incorrectly, used interchangeably.

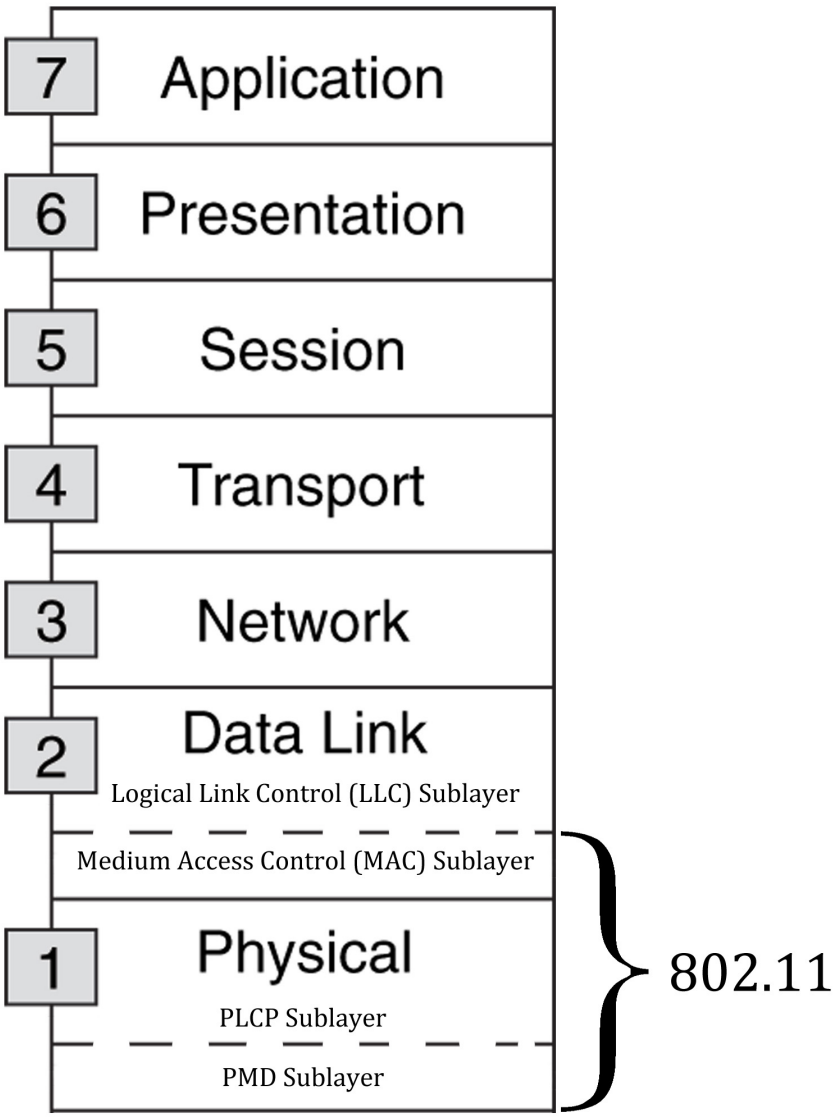

Figure 2.1: The OSI reference model

2.1.2 Operating Modes: Ad-hoc and Infrastructure

There are two operating modes defined in 802.11: ad-hoc and infrastructure mode. In Ad-hoc mode each STA communicates directly with another STA without the use of an AP. In infrastructure mode on the other hand there is a dedicated AP that is providing services to all the STAs in the given network. When in infrastructure mode a STA can only communicate with the AP directly, the AP will in turn relay

the traffic to the correct receiver STA on the local network or to another destination on the Internet. The latter approach is the one most commonly used, and it is the one found in WrT. For this reason infrastructure mode will be the only operating mode in focus in this thesis.

2.2 The 802.11 MAC Frame

The 802.11 standard [6] defines three different frame types; control, management and data frames. All information and signaling between STAs and APs in 802.11 networks is sent by using one or more of these frame types. The frames are sent on the MAC layer, and more information on each type as well as some of the subtypes will follow in later in this chapter.

2.2.1 The Frame Structure

Every MAC frame, regardless of the type has the structure shown in Figure 2.2. The first three fields (`Frame Control`, `Duration/ID`, and `Address 1`) and the last field (`frame check sequence (FCS)`) constitute the minimal frame format and are present in all frames. The other parts are only present in certain frame types and subtypes [6].

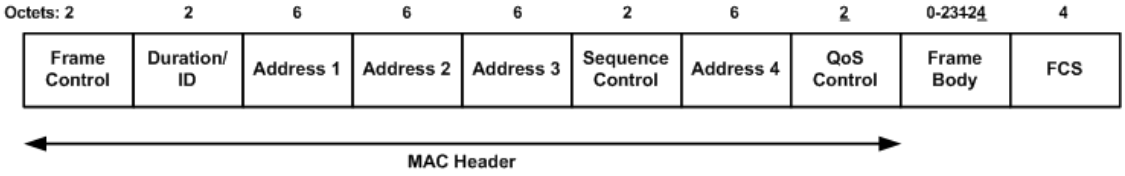

Figure 2.2: The general 802.11 frame [6]

Frame Control The `Frame Control` field consists of several subfields, some more important for the work done in this thesis than others, and these important fields will be explained now. See Figure 2.3 for the format of the `Frame Control` field.

The `Type` subfield identify whether the frames is a management, control or data frame, the `Subtype` field further identifies the subtype. In data frames, the most significant bit (MSB) of the `Subtype` field, b7, is defined as the QoS subfield. A value of 1 here indicates that the QoS field is present in the MAC frame [6].

The `Power Management` field (denoted PwrMgt in the figure) is used to indicate the power management mode of a STA, more details will be presented in the section on power management, Section 2.4. For a complete overview of the `Frame Control` field consult Section 7.1.2 in [6].

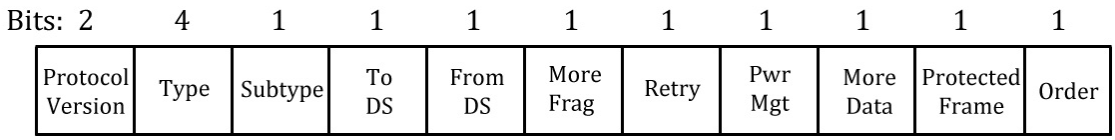

Figure 2.3: Format of the `Frame Control` field

The Remaining Fields `Duration/ID` is the first field after `Frame Control`, and in most cases it contains the calculated value from a STAs perspective on how long it will take until the frame is transmitted. An exception to this is in PS-Poll frames were this field contains the association ID (AID) of the STA. This is part of the power management protocol and more information will be presented in Section 2.4.

The four addresses are MAC addresses and they are called source address (SA), destination address (DA), transmitting STA address (TA), and receiving STA address (RA) respectively. In our case when a STA sends frames to an AP the different fields will have the following values. SA = TA while RA = the AP's address and DA is the address to the final destination somewhere on the local area network (LAN).

`Sequence Control` is the field including a sequence number for each frame delivered between two stations. This field has traditionally been the most important field in detection of MAC spoofing attacks, see for instance [1], but this field will not play an important part in the detection algorithm presented in this thesis.

The `QoS Control` field is specific for 802.11e and it is only present if the AP and STA supports it. This field is 16 bits long. The first four bits make up the `traffic identifier (TID)` subfield, and is used to determine the QoS class, or TID class in the 802.11 frame. This allows for a total of 8 different TID classes. If QoS is not in use by the active application, Differentiated Services Code Point (DSCP) or a similar service the default priority value of 0 used.

The actual data or payload is inserted in the `Frame Body` field, and it is variable. The maximum size of the payload is 2312 bytes per frame.

Finally the `FCS` field is added for error control, it consists of a cyclic redundancy check (CRC) [6].

2.2.2 Frame Types

The three different types of frames in the 802.11 standard are control, data and management. The control frames assist in the delivery of data frames, but do not carry any payload itself. They are used for time-critical signaling and examples of such frames is the acknowledgment (ACK) frame which is used to acknowledge that another frame is received and the PS-Poll frame which is used in power management. Another pair of well known control frames is the frames used in the RTS/CTS

protocol[1] to avoid frame collisions. Note that the RTS/CTS protocol is optional and some networks do not use it due to the extra traffic overhead it introduces. Control frames do not include the `Sequence Control` field in their frame body.

Data frames carry the actual payload, this is done by encapsulating packets from a higher layer in the OSI model. The payload has a maximum size of 2312 bytes, and as a result larger frames must be fragmented. A special version of the data frame is the Null Data frame. It does not contain any payload, and as will be explained later, it is used for several different things depending on the implementation.

Management frames are used to establish and maintain the wireless link between an AP and a STA. This is also signaling, but compared to the control frames management frames are not as time-critical. The subtypes for management frames include specific frames for association, disassociation, authentication and deauthentication.

2.3 Detailed Information on Relevant Frames

Some frames are more relevant than others in order to understand the work done in this thesis as they are used in the fingerprinting algorithm developed. These frames all share the common trait that their values and usage differ depending on the implementation and equipment used. A more detailed description of these frames will now follow.

2.3.1 Beacon Frame

The Beacon frame is a management frame (type 00) with subtype 1000. Beacon frames are periodically transmitted by the AP. It is used to announce the network to new clients and to inform STAs about several different parameters and settings regarding the network. The most relevant parts of the Beacon frame for the work done in this thesis will now be presented. For a full overview the reader is referred to Section 7.2.3.1 in [6].

The `Timestamp` field contains the value of the timing synchronization function (TSF) timer of the AP and is used to inform STAs about the correct time to enable synchronization. Each STA maintain a TSF timer counting in increments of microseconds. This is a 64-bit modulus timer [6].

The second field in the Beacon frame is the `Beacon interval` field. This field represents the number of time units between target beacon transmission times [6].

[1] RTS/CTS is a protocol where a node wishing to send data initiates the process by sending a request to send (RTS) frame. The destination node replies with a clear to send (CTS) frame. Any other node receiving the RTS or CTS frame should refrain from sending data for a given time (solving the hidden node problem).

The `service set identifier (SSID)` field is the fourth field in the frame body, it is an information element, and it indicates the identity of the network (the name). In our case the SSID equals *Wireless Trondheim*.

The last field to be highlighted in this section is the `traffic indication map (TIM)` field. This is also an information element and the full details of its structure can be found in Section 7.3.2.6 in [6]. The TIM element contains a field called `Partial Virtual Bitmap`, this field is 1-251 octets long and each bit corresponds to a STA's AID. More details regarding its usage is given in power management section.

2.3.2 Association Request

The association request frame is a management frame (type 00) with subtype 0000. This is an important frame in the authentication-association procedure and it contains a lot of configuration information from the STA that is connecting to the network. The frame contains the different fields and information elements as shown in Table 2.1. The `Capability` field is two octets long a and consists of a number of subfields that are used to indicate requested or advertised optional capabilities [6, Section 7.3.1.4]. The `Listen interval` field is as described in Section 2.4. The purpose of the next fields and information elements is pretty self-explanatory by their name in Table 2.1, if you still want a more detailed description consult Section 7.3 in [6].

The three first elements of the frame are called fields and they are always present. The next elements are called information elements, and do not have to be present under every condition. An example is the `Extended Supported Rates` element which must be present if the STA supports more than eight rates, but is optional otherwise. There might also be zero, one or several vendor specific elements present at the end of the frame. For a complete overview it is once again referred to Section 7.3 in [6]. A lot of differences between implementations and equipment are found in the association request frame, and these differences provide an opportunity to fingerprint as will be demonstrated later.

2.3.3 PS-Poll

The PS-Poll frame is a control frame (type 01) with subtype 1010. The format of the PS-Poll frame can be seen in Figure 2.4. It has no payload, and the `Duration/ID` field contains the AID of the STA. This is a 16-bit identifier (ID) given to the STA by the AP in the association response during the initial handshake. The PS-Poll frame is used in power management as described in Section 2.4.

Order	Information
1	Capability
2	Listen interval
3	SSID
4	Supported rates
5	Extended Supported Rates
6	Power Capability
7	Supported Channels
8	RSN
9	QoS Capability
Last	Vendor Specific

Table 2.1: Association Request frame body [6]

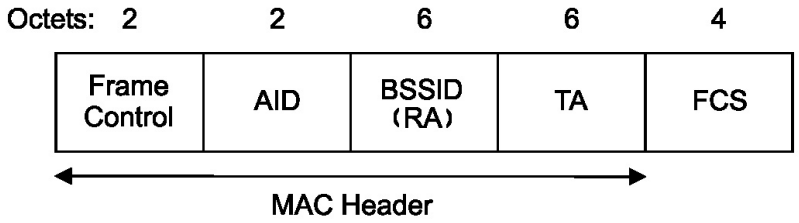

Figure 2.4: The PS-Poll frame body [6]

2.3.4 Null (no data) and QoS Null (no data)

The Null (no data) and QoS Null (no data) frames are both data frames (type 10) with the subtype 0100 and 1100 respectively. For simplicity the parenthesis will be dropped and these frames will be referred to as Null Data frames throughout the report. The frame body of the Null Data frame can be seen in Figure 2.5. There is no payload and the `QoS` field is only present in QoS Null Data frames, this is true regardless if QoS is activly used or not (default TID = 0 in use).

The 802.11 standard does not specify the use of the Null Data frames. Nevertheless they are used extensively, and for many different purposes. In [4] it is stated that Null Data frames have three main usages; association keep alive, changing power management mode and channel scanning. More details of the specific use of Null Data frames in these scenarios will be presented in Section 3.4.1.

2.4 Power Management

Power management enables STAs to save power by turning off their receivers at certain times without the risk of not receiving frames by doing so. Most mobile devices use power management as reduced power consumption is a desired trait.

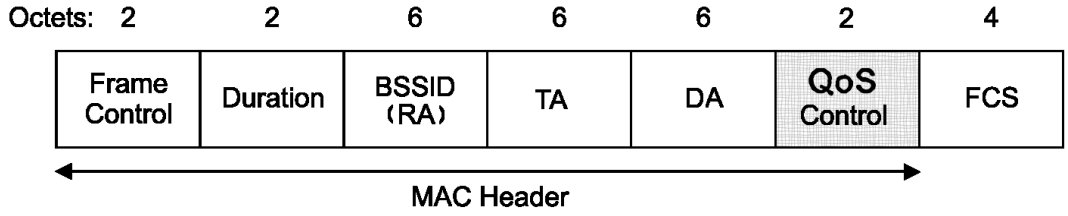

Figure 2.5: The Null Data frame body

2.4.1 Background

During the association a STA is assigned a 16-bit ID called AID. The value assigned as AID is in the range 1-2007. Each STA have its own `ListenInterval` that indicates the maximum number of beacon intervals the STA might be dozing before it awakes and listens for Beacon frames.

Recall that Beacon frames contain the information element `TIM` which again contains the field `Partial Virtual Bitmap`. The `Partial Virtual Bitmap` contains up to 251 octets (2008 bits) and each bit corresponds to a AID of a STA. If bit x equals 1 then the AP has buffered one or more MAC service data units (MSDUs) for the STA with AID x.

2.4.2 The Frame Exchange

STAs changing power management mode shall inform the AP of this fact using the `Power Management` subfield within the `Frame Control` field of transmitted frames. This subfield is only one single bit and the value indicates the mode in which the STA will be after the successful completion of the frame exchange sequence.

The standard does not specify which frames that should be used for this purpose, but it does provide the requirement that a change must be done via a frame exchange that includes an acknowledgment from the AP. In other words, using an ACK would not suffice, but using an Null Data frame would be good as the AP would ACK the Null Data frame. Null Data frames are often used as they are very lightweight.

2.4.3 How Power Management Works

A STA can be in two different power states, *Awake* and *Doze*. While in *Awake* the STA is fully powered, when in *Doze* on the other hand the STA is not able to transmit or receive anything and consumes very little power.

The state of a STA is determined by the STA's power management mode. There are two different modes, active mode (AM) and power save (PS) mode. A value of 1 in the `Power Management` subfield indicate PS mode while 0 indicates AM. In AM

the STA is in the *Awake* state and may receive frames at any time. In PS mode the STA is in the *Doze* state and shall only enter the *Awake* state to receive selected Beacon frames based upon the `ListenInterval`.

If the TIM element in the Beacon frame have a value of 1 in the bit map corresponding to the STA's AID the AP has buffered one or more MSDUs while the STA was in the *Doze* state. The STA then sends a PS-Poll frame to the AP, and the AP responds by transmitting the buffered MSDUs to the STA. This is done even if the STA is still in PS mode.

For a complete overview regarding power management in 802.11 consult Section 11.2 in [6].

2.5 Radiotap Header

The radiotap header is not a header that is actually sent, but it is created on the receiver side and is a mechanism to supply additional information about frames [14]. The radiotap header is included in the captures done in the experiments as it contains some important information needed for the fingerprinting.

2.5.1 MAC Timestamp

Timestamps are available through the `TSFT` field in the radiotap header. The `TSFT` field, known as `radiotap.mactime` in Wireshark, is defined as follows: *Value in microseconds of the MAC's 64-bit 802.11 TSF timer when the first bit of the MAC protocol data unit (MPDU) arrived at the MAC. For received frames only* [15]. This field will be used in the algorithm developed in this thesis.

2.5.2 Data Rate

The datarate used to transmit a packet is not contained in the MAC frame, but it is also available in the radiotap header. This information is available in the `Rate` field, also known as `radiotap.datarate` in Wireshark. The definition of this field is: *TX/RX data rate in units of 500 Kbps.*

The rate information is actually extracted from the PHY, more specifically from the physical layer convergence procedure (PLCP) header in the PLCP sublayer. The PLCP header varies depending on the type of PHY specification used. For more information consult the 802.11 standard under the PHY technique in use, see for instance Section 19.3 in [6] for details regarding the PLCP sublayer in extended rate PHY (ERP), commonly known as 802.11g.

2.6 Relevant Attacks on 802.11 Networks

This chapter will present different types of attacks that are relevant in order to perform an identity theft in form of a MAC spoofing attack. More precisely the chapter will focus on how such an attack is performed in an open 802.11 network with a captive portal, i.e. the network type found in WrT. For a more general overview and introduction to attacks on 802.11 networks see for instance [10].

2.6.1 Eavesdropping and Traffic Analysis

Eavesdropping and traffic analysis is a group of passive attacks. In a passive attack the attacker does not interact with the network at all. He does not generate any traffic, but passively listens to the packets available "in the air" at his physical location. As a result of not transmitting anything these attacks are almost impossible to detect, in addition they are also simple to perform as one just need to receive packets.

In the eavesdropping attack the attacker typically uses some sniffing tool[2] to capture all the packets available in his geographical position. The challenging part for the attacker is then to be physically close to the AP and/or STAs in question in order to receive the packets. Gathering packets through a sniffing tool constitutes a successful eavesdropping attack, and in an unencrypted network this would give the attacker a complete overview of what the victim(s) are communicating.

Traffic analysis is the process of deducing information from the traffic patterns found by observing the number and frequency of packets transmitted as well as their length and type. This is typically a lot more complex attack than simple eavesdropping and requires detailed knowledge about the protocols in use in order to draw some conclusions.

Eavesdropping and traffic analysis are often the starting point from an attacker's perspective in order to gather information about specific clients and the network in general. These attacks will almost never be the end goal itself, rather they will be a stepping-stone to enable other, more complex attacks. This holds true for the MAC spoofing attack which will be described later in this section.

2.6.2 Masquerading: MAC Spoofing

As noted earlier a user of WrT's network must have his MAC address whitelisted before the he is able to access anything except the login-page. The whitelisting is done in the Nomadix gateway. An overview of the network topology depicting the

[2]Wireshark, Airodump, Kismet etc.

gateway and other important elements will be presented in Chapter 3. An easy way to fool this system is by performing a MAC spoofing attack.

Overview of the MAC Spoofing Attack The theory of the MAC spoofing attack is that the attacker masquerades as an legitimate client, one who already has his MAC address whitelisted. Thereby the attacker gains access to the network. As the name implies the masquerading is done by spoofing the MAC address of the legitimate client. This information can easily be obtained through eavesdropping. Note that the attacker will only have access to the network as long as the legitimate user is whitelisted.

There exist several variations of this attack. They differ in the way the victim is treated, whether the attacker tries to avoid detection, and as a result the complexity of performing the attack. In the remaining of this section five different versions will be briefly explained without going into the practical details of performing each of the attacks. The categorizations and explanations is based on the paper written by Xia [8] as well as a thorough overview of these attacks described by Idland in [1] specifically for the network type found in WrT.

MAC Freeloader Attack This is the simplest variation of the attack. It does not require much skill, and it does not try to avoid any form of detection. The attack consists of spoofing the MAC address of a legitimate client and nothing more. In this case both the attacker and victim will be connected to the network at the same time, and as a result both STAs will receive each other's traffic. It is pointed out in [8] that this behavior should cause problems in the Transmission Control Protocol (TCP) layer. More specifically it should result in a termination of the ongoing TCP connections, thus making it very difficult for the attacker to use TCP. This is usually not the case in practice, and the same paper points out that modern firewalls block TCP traffic not originating from the client and thereby aiding the attacker in this specific case.

QoS Optimized MAC Freeloader Attack In the QoS optimized version the attacker tries to avoid detection based on simple SQN analysis. This is done by changing the 802.11 QoS priorities to some priority not currently in use by the legitimate client. Each QoS class has its own sequence counter. By using an previous unused class the attacker will obtain his own counter and fool some systems based on plain SQN analysis. This variation is described here for completeness as WrT runs a QoS enabled network, but it will not play a significant role in this thesis. See [1] for an algorithm specifically designed to detect this attack.

Wait-for-Availability Attack This attack is very similar to the freeloader attack, but there is one important difference. In this case the attacker waits until there is no longer any traffic going to or from the victim's STA. Then, assuming the victim has left; the attacker spoofs the address and tries to connect. In WrT a MAC address stays whitelisted for 20 minutes after the traffic has ceased regardless if the client deauthenticated or just stopped transmitting. This is the most relevant version of the MAC spoofing attack as it is easy to perform and very hard to detect with existing methods based on two STAs transmitting at the same time.

Session Hijacking — Deauthenticate Attack In this variation the attacker first spoofs the AP's MAC address in order to send fake deauthenticate messages to the victim. The victim's STA believes that the deauthenticate is legit, and as a result it terminates its association with the AP. The attacker can then spoof the victims MAC and have the session for himself. To prevent the victim from reconnecting the deauthenticate procedure will have to be repeated frequently.

Session Hijacking — Channel Switch Attack A recent variation of the session hijacking, called channel switch attack, is described in [16]. In this version the attacker does not send fake deauthenticate messages; instead he sends a channel switch announcement element. A correctly crafted element would result in the victim changing channel (to an invalid one) immediately, and stay on that channel for up to 255 beacon intervals before switching back. The authors report that a denial of service (DoS) effect of up to one minute can be achieved with a single message.

2.6.3 Man in the Middle

The man in the middle (MITM) attack is a more complex attack than the MAC spoofing attack, but it turns out that the added complexity might be worth it as this attack is also much harder to detect. The attack can be described as follows.

The attacker sets up a rogue AP[3] and waits for a legitimate client to connect. All the data is then relayed to the real AP. The real AP then interpret the rogue AP as the legitimate client [10].

The victim believes he is connected to the real network and continues as usual. The network also believes that everything is correct as the attacker have full control over the frames sent to the real AP. Having full control over all the frames sent allows the attacker to inject his own frames into the stream from the legitimate

[3]A rogue AP is an AP controlled by the attacker. It can be a physical AP or created in software running on a laptop. It masquerades as a real AP of the target network — preferably undistinguishable from a regular clients view.

client and modify them in such a way that sequence numbers and other parameters behave correctly. As noted in the beginning of this subsection the MITM attack is a lot more complex than the previously mentioned attacks, but with automated software this could be a very viable attack, even for a novice attacker.

It should also be noted that a channel switch attack could be used to trick a STA to change from a legitimate AP to a rogue one. This technique allows the attacker to actively acquire clients and not just wait for a client to connect to his rogue AP by chance. At the same time the sending of channel switch announcement elements makes detection easier.

2.7 MAC Spoofing Prevention

2.7.1 Robust Security Network

A logical solution to significantly reduce the risks associated with MAC spoofing attacks would be to not use the MAC address as a vital part in the identification process. Using other methods to distinguish and identify users could be implemented, for instance with the use of Robust Security Network (RSN) [6]. RSN is not used in WrT as it requires keys and certificates, and is therefore regarded as too much hassle for the average user that just wants access to the Internet while sitting on a café in downtown Trondheim.

2.7.2 Active Methods

Active methods are a class of methods that does not require the setup of keys or certificates, thus reducing the hassle for the user. What these methods do require is an active STA. Several active countermeasures to prevent the effectiveness of MAC spoofing attacks exist. An good example that demonstrates this class is the active SSL window technique mentioned in Section 1.5.2 and explained in [8]. The downside with this technique is that it requires the STA to run two browser windows at the same time, a requirement that is currently not supported by a host of smartphones and other Wi-Fi equipment.

Keeping in mind WrT's goal of having an easy to connect to network that is open for every type of Wi-Fi equipment it is obvious that this puts some constraints on what one can do regarding security. To the author's knowledge no active method exists that can prevent MAC spoofing attacks while at the same time upholding WrT's goals. As a result the focus has in the last years been, and will continue in this thesis to be on detection.

2.7.3 Whitelist Flushing

A technique worth mentioning, and that could be used is something called a log-off function. WrT have the possibility to enable this functionality, but it is currently not in use. The idea is to have a button on a webpage or some known Uniform Resource Locator (URL) one can visit to log-off when you are done. What happens then is that your MAC address is flushed from the whitelist immediately. This would effectively prevent the wait-for-availability attack which is the main attack in question, but it would do nothing to prevent the other types of attacks.

Giving each user the responsibility of logging off would probably not yield the best results as most users would forget to do it, or simply not care whether they flushed the whitelist or not. An idea would then be to flush the whitelist automatically on a deauthenticate message, thereby ensuring that it is done when the user logs off. The problem with this approach is that some devices, for instance the Apple iPhone sends deauthenticate messages whenever it enters sleep mode [1]. The user would then be required to enter his credentials in the captive portal again after awaking the phone. This could happen quite frequently and the additional hassle would probably not be worth the increased security.

2.8 Fingerprinting

Fingerprinting is the act of collecting externally observable characteristics from a specific source in order to identify it, i.e. develop a fingerprint of that source. A good fingerprint resembles fingerprints in humans in the way that they are unique or almost unique for each individual. In the case of WrT the target of the fingerprinting, or the fingerprintee, is typically the STA.

Fingerprints exist as a consequence of implementation differences between different devices and user specific options. These differences are often reflected in the type of traffic generated by a given STA. Fingerprints for a known device with known settings can often be pre-determined. The task is therefore often to determine which kind of STA we are dealing with by observing and potentially interacting with it, and then comparing the observed data with known fingerprints.

2.8.1 Active versus Passive

Fingerprinting can be done actively or passively. Passive fingerprinting is the most straightforward case, and it does not involve interaction with the fingerprintee. In this case the fingerprinting program acts as an observer without sending any requests or queries. The fingerprinting technique used in this thesis is of the passive sort.

On the other hand, an active fingerprinting program is one that is sending some kind of request to the fingerprintee in order to obtain additional information. The idea is that by being able to create the request one might get more relevant information from the target than what is acquired through merely observing. An example of active fingerprinting is found in the feature called *OS Detection* in the popular network scanner Nmap [17]. It is stated on their web-page that this feature is based on TCP/IP stack fingerprinting where Nmap sends a series of specially crafted TCP and User Datagram Protocol (UDP) packets to the remote host and examines practically every bit in the responses [18].

An active fingerprinting system is often more complex than a passive one as it not only observe (receive packets), but also transmit packets. Typically an active system requires more hardware in order to transmit as well. In some cases the extra cost of complexity and hardware might be acceptable if the active techniques increase the success rate substantially. Other times a passive system would have the same success rate as an active one, and thus it would not be worth the extra costs.

2.8.2 Sources for Fingerprinting

There are many different sources for fingerprinting and this section will present some of the more popular and relevant sources of fingerprinting in 802.11 networks.

User Users can be a good source for fingerprinting as they usually have some preferences reflected in settings and options that differentiate them from other users. In addition to different settings the behavior pattern of a user can be used to fingerprint as well. Examples of the latter could be the duration of an association, type of protocols used and first web-page loaded (start-up page).

Web Browser Different browsers generate different requests and responses during normal use. With the addition of plug-ins and user specific settings the entropy of a browser is reported to be over 18 bits [19]. The author in [19] created a fingerprinting algorithm and showed that for browsers which support Flash or Java 94.2% had a unique fingerprint. This demonstrates that browser fingerprinting can be very viable.

TCP/IP Stack Some of the parameters within the TCP are not specified by the standard. As a result implementations differ in how these parameters are calculated and used. The set of these parameters include the initial value of the `TTL` field, the `Window Size` field and the `DF` bit to name a few [20, page 229]. There exist several programs that fingerprint on the TCP/IP stack, and amongst the more popular is the already mentioned active program Nmap [17] and the passive program p0f [21].

MAC Layer As with the TCP there are several parameters and options that are not unambiguously specified in the 802.11 standard. This gives room for different implementations which again provides us with a fingerprinting possibility. Two examples of implementation differences in the 802.11 standard is the use of Null Data frames and how the `Duration` field is calculated. MAC layer properties is the source of choice in this thesis. A thorough explanation of how the fingerprinting on the MAC layer is performed, as well as the rationale behind choosing MAC layer fingerprinting as the preferred source will be given in Chapter 3.

2.9 Intrusion Detection Systems

2.9.1 Introduction to Intrusion Detection Systems

An IDS is a system designed to detect attacks through monitoring and analyzing live behavior for abnormalities. These systems often require some additional hardware in order to monitor and potentially interact with clients. This hardware could be standard off the shelf equipment as a regular wireless card or it could be very specialized equipment.

IDSs are similar to intrusion prevention systems (IPSs), but an IPS has its main focus on preventing and blocking an intrusion before it is successful. IDSs on the other hand will only detect an attack after it is executed, but hopefully very soon after.

The ideal IDS will detect every attack and never flag normal behavior as suspicious. A real world IDS will never be as good as the ideal, but it should be designed with the goal of approximating the ideal IDS in terms of reliability.

There is no one preferred method of detection, but it is a common understanding that the combination of several techniques will increase the probability of the IDS to succeed. The most relevant detection techniques in order to detect a MAC spoofing attack will be presented below.

2.9.2 Active versus Passive

IDSs can be grouped in several ways; one of them is how they interact with the environment, in our case the network. An IDS can either be passive or active. Being passive implies that the IDS does not interact with the network or STAs at all, the system is only a passive observer. On the contrary an active IDS does interact with the network, STAs or both in an attempt to elicit abnormal behavior. An example of active behavior is a system that sends out a request and analyzes the response. The request could have been crafted in such a way that a specific

response would be expected from legitimate clients while another response might be expected from clients with irregular settings.

This is parallel to the active versus passive discussion regarding fingerprinting programs found in Section 2.8.1, both regarding the behaviour and tradeoff between complexity and success rate.

2.9.3 False Positives and False Negatives

False positives and false negatives are two important terms used in relation to IDSs. The first term, false positive, is used to denote a situation where the system has detected suspicious behavior (flagged an attack) when there in fact is no such attack. The last term, false negative, is used to imply that an actual attack was not detected by the IDS. Ensuring low rates of both false positives and false negatives is paramount for any IDS in order to be reliable and useful in real situation.

Another term that is closely related to the term false negatives is the success rate of an IDS. The success rate is the percentage of attacks that is detected. In other words the success rate is the compliment of the false positive rate, and they together add up to 100% (all the attacks).

2.9.4 Statistical Anomaly- and Signature-based Detection

There are two main types of detection techniques, statistical anomaly-based and signature-based. An IDS which utilizes statistical anomaly-based detection will typically have a baseline of what behavior that is considered normal. Activity on the network is monitored and then analyzed in accordance with the baseline parameters. If the behavior does not fall within what is defined as normal the user or STA will be flagged as suspicious.

In signature-based detection the IDS analyzes the activity, searching for specific attack patterns known as signatures. Once a known attack pattern is recognized the user or STA will be flagged as suspicious. This type of detection resembles the one used in anti-virus software and a similar problem arises; attackers will adapt and change their methods so that they no longer fit the attack signature, and thus the IDS requires frequent updates of signatures [22].

2.10 Relevant Detection Techniques

This section will present several classes of detection techniques that have been used in wireless IDSs. Some of these techniques have proven to be efficient in the relevant environment as described in Chapter 1 while others not. The reason for success or lack therof will be explained.

2.10.1 MAC SQN Analysis

MAC SQN analysis is a technique based on how the sequence numbers operate in the 802.11 standard. Each STA have its own SQN counter that increase monotonically. During a MAC freeloader attack, described in Section 2.6.2, there will be two STAs each transmitting with the same MAC address. These STAs will each have their own SQN counters. From the IDS's viewpoint the numbers will therefore not increase by one for each frame delivered from that MAC address, but rather jump back and forth as a result of following two independent counters. See Figure 2.6 for an example of how the SQN could behave seen from an observer's (IDS) perspective.

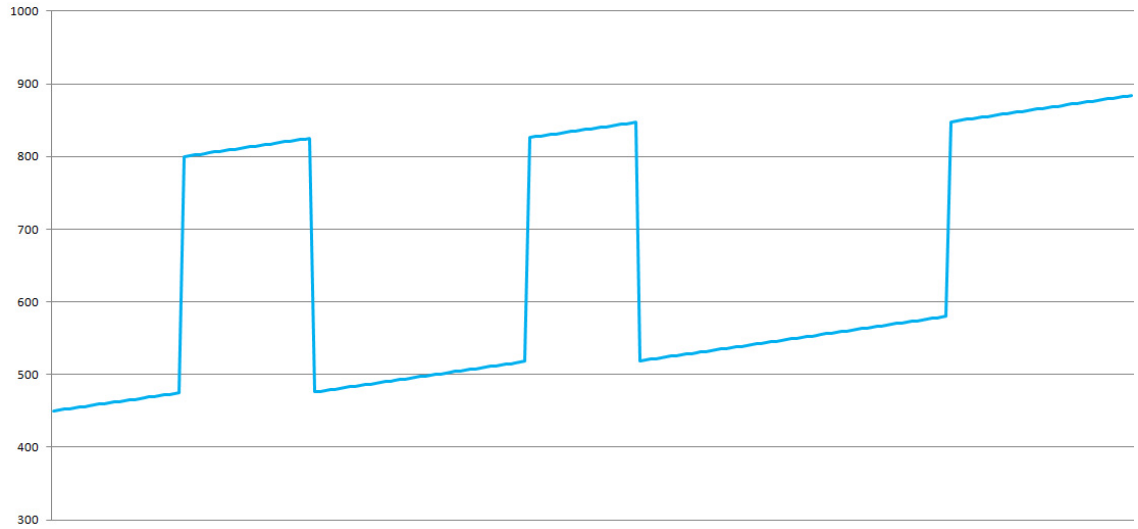

Figure 2.6: Observed SQNs during a freeloader attack

This detection mechanism was presented in the paper *Sequence Number-Based MAC Address Spoof Detection* [12]. It later formed the basis for an proof-of-concept IDS developed specifically for WrT by Holgernes and Pedersen [23] and later improved by Idland [1]. This detection technique plays an important role in detecting MAC spoofing attacks, but it is limited to the types of attacks where attacker and victim transmit at the same time. As this thesis focuses on detecting the attacks where the attacker and victim are not connected at the same time the MAC SQN analysis will not be used.

2.10.2 RSS Monitoring

Received signal strength (RSS) is a value measured by the receiving network interface controller (NIC), indicating the signal strength at the receiver. In this detection technique the IDS analyzes the RSS measurements, looking for abnormalities. A scenario with oscillating values originating from the same MAC address within a short timeframe would typically be flagged as suspicious.

A major strength with this technique is that it works regardless of protocol, encryption or any other settings as the measurements are done on the physical layer. It is also very hard for an attacker to spoof the value as the value itself is not sent, but measured at the receiving party [5, 24].

As mentioned oscillating values in the RSS would typically be flagged as suspicious. The rationale behind this decision is that different values is an indication that there are in fact two STAs located in different positions and sending with the same MAC address. In a static network, as within an office premises, this would be true as demonstrated in [5]. As mentioned in the related works section Pedersen looked into how this technique would perform in an urban and dynamic environment as WrT. He concluded that RSS monitoring as detection technique in WrT and similar networks are in fact unreliable [2].

2.10.3 RTS/CTS Roundtrip Analysis

Two of the control frames in 802.11 are the RTS and CTS frames. They are part of an optional protocol used to avoid collisions due to the hidden node problem mentioned earlier. The detection technique is based on analyzing the RTT of the RTS/CTS message exchange. In [25] the authors are able to estimate the distance from the IDS to a given STA with a meters accuracy.

In this case, as in the case of RSS monitoring, oscillating values could indicate an attack, i.e. two different STAs. The downside is once again that the experiment in [25] was performed with a stationary STA, and thus are not applicable to our network for the same reasons as with RSS monitoring. In addition several networks, including WrT, do not have the RTS/CTS protocol enabled as it results in increased overhead traffic.

2.10.4 Protocol Specific Analysis

This technique is based on analyzing the packets originating from a MAC address, looking for inconsistencies or irregularities when taking into account the protocol standard. Holgernes used this detection technique with the 802.11 protocol in his master's thesis [13] in order to detect the freeload attack. Specifically he analyzed how the 802.11 states operate and which management frames that are invalid in a given state. Not conforming to the standard would be regarded as suspicious. The theory is that two STAs that follow the standard with the same MAC address would appear to be one STA not following the standard.

Contrary to the previous methods based on physical measurements this technique is not automatically deteriorated in a mobile environment, but it is very dependent on having a low drop-rate. If a single management frame is dropped it could cause

the state held by the IDS to be different from the actual state in the STA, the next management frame received would then probably result in a false negative or a false positive.

The sending of management frames is not an unspoofable property as it is defined in the drivers of the NIC. Nevertheless Holgernes argues that it would be a complex task to edit these drivers in order to stop them from sending certain management frames. Regardless this technique is also of little use when regarding the wait-for-availability attack as the legitimate client then is already deauthenticated and will not transmit any more frames.

2.10.5 Behavior Monitoring through Fingerprinting

Behavior monitoring through fingerprinting as a detection technique requires some set-up time. First the behavior of the STAs in a network is monitored, and based on several specific traits, a fingerprint of the STAs is developed. Recall fingerprinting from Section 2.8. The creation of the fingerprints is under the assumption that no attack is currently ongoing, thus the STAs are uniquely identified by the MAC address.

Then, if an attack is performed after the initial set-up the fingerprint should change as a new STA (the attacker) is sending frames with MAC address of the first (legitimate) STA. This sudden change in a fingerprint for a given MAC address would then indicate that an attack has been performed.

The fingerprint should be based on several different properties and they should not be easy to spoof. In this thesis such a fingerprint is developed on the basis of heterogeneity found in the 802.11 standard. As is turns out there are a lot of differences to be utilized in a fingerprinting algorithm, and several of them are not easily spoofable as they are a result of the drivers running the NIC.

One drawback with this technique is that two identical STAs probably would have an identical fingerprint. A major strength on the other hand is that this technique in theory should work very well against the wait-for-availability and the session hijacking attack. The rest of this thesis will be focused on developing an algorithm for detecting these attacks through fingerprinting and testing it in a real network.

Chapter 3

The Fingerprinting Algorithm

This chapter will present the background for why fingerprinting can be used as a viable detection method and a detailed description of how the fingerprinting algorithm is developed. It will start by defining the threat model in terms of network and attacker with a discussion of relevant attacks. Then a rationale for fingerprinting as the preferred detection method is given before a discussion of potential fingerprinting properties. Finally, details of how the algorithm was developed and how it works is presented, this includes pseudocode and literal descriptions.

3.1 Threat Model

The treat model includes descriptions of the network in question and a generic attacker. The attacker is defined in terms of motive, skills, know-how and equipment, while the network is described by its relevant equipment and software. These descriptions are important in order to understand the decisions made in this thesis.

3.1.1 The Network

The network in question is WrT's network. An simplified overview depicting the relevant components and the network's topology can be seen in Figure 3.1. On the wireless side WrT hosts an open 802.11 network with WMM (QoS) enabled. Both 802.11a, 802.11b and 802.11g are supported. WrT do not have regular APs, but instead uses LAPs. These LAPs are minimalistic APs and all the setup and settings is controlled from the WLC. The captive portal functionality is implemented in the Nomadix gateway. No IDS or IPS is currently used in the network.

The entire network is naturally a lot more complex and includes several LAPs, WLCs and so on. For simplicity and readability these entities are not included in the figure. What is included on the other hand is a legitimate STA, a attacker STA and the computer used as monitor in the experiments (marked IDS server). It should

be noted that the actual attacks and other experiments were performed on the real WrT network. In the threat model the wired network in considered secure, whereas the wireless is not. The following can therefore be stated about the network:

Attacks on the network will only be considered on the LAPs.

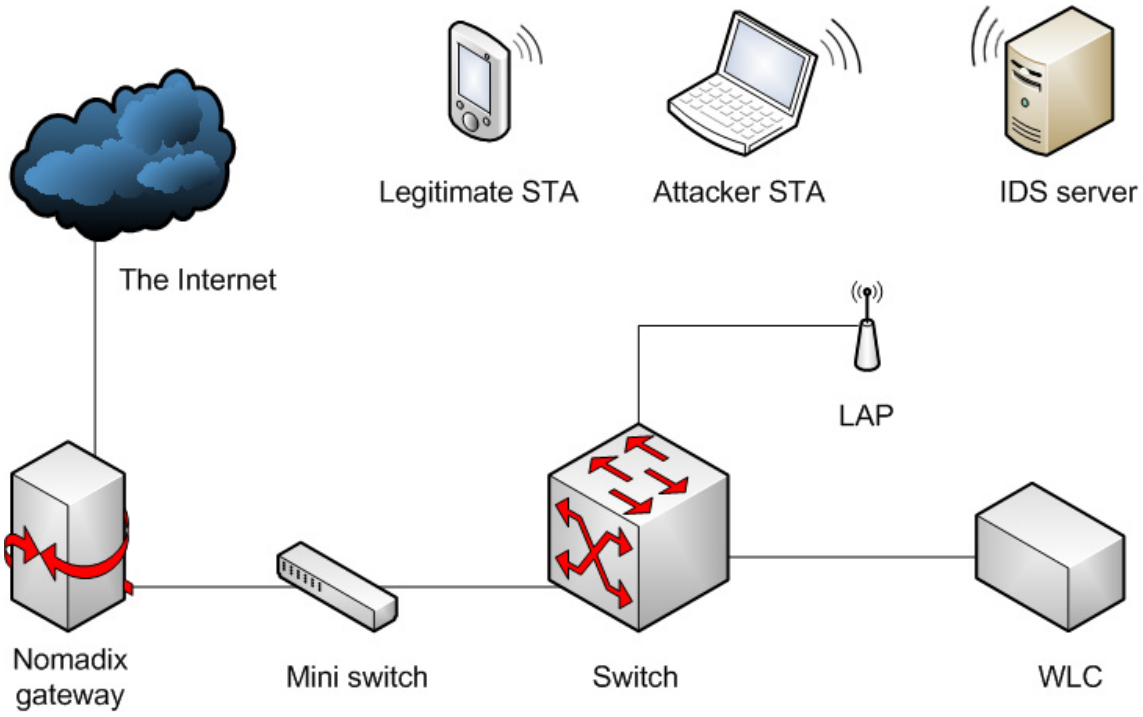

Figure 3.1: Overview of Wireless Trondheim's network [1]

3.1.2 The Attacker

Motive There are a lot of possible specific motives for performing an identity theft. One of the most serious cases is the case when the attacker wishes to perform some kind of illegal activity masquerading as an innocent client, and this is the case WrT is concerned about. Browsing of sexual abuse images of children is a typical example of the latter. Generally the motive of the attacker is defined as:

The attacker's motive is to gain access the network and the Internet masquerading as someone else.

Skills WrT want host a network that is easy and convenient to connect to and security features should not interfere with this. As a result of not proclaiming a bulletproof network a highly skilled attacker is not considered. What is considered is an attacker that is able to perform the different masquerading attacks described in Section 2.6.2 as well as passively listening to the network traffic. It is also fair

to assume that he has knowledge about SQN analysis and other popular detection methods. His skills are defined as follows:

The attacker has the skills to perform the five different versions of the MAC spoofing attack. In addition he have a thorough understanding of the basics behind the 802.11 protocol, IDSs and commonly used detection methods.

Equipment The equipment available should match the motivation and skills of the attacker. In order to limit the scope of this thesis advanced equipment is assumed to not be available. The attackers equipment is defined as:

The attacker has a laptop with a wireless NIC and have access to all relevant software. Other physical equipment is not considered available.

3.2 Discussion of Possible Attacks

The attacker with limited skills as defined in the threat model can perform several different attacks; some that will be considered, while others not. The current section presents a discussion of the different attacks and whether they are considered in this thesis. This is mainly done in order to specify the scope of the work done in this thesis.

3.2.1 Eavesdropping and Traffic Analysis

Eavesdropping and traffic analysis is often the starting point for other attacks, specifically this is true for the MAC spoofing attack. Naturally these types of attack will be considered.

3.2.2 MAC Spoofing

In the threat model the attacker is defined to have the skills to perform all of the five different types of MAC spoofing attacks. Nevertheless not all of them will be considered when developing the detection techniques in this thesis. Idland reports good results in detecting the freeloader type with or without QoS optimization [1]. As this is a continuation of his work those attack types will not be considered.

The relevant type of MAC spoofing attack is the ones were the attacker and victim are *not* connected simultaneously. This includes the three remaining types. The main focus will be on the wait-for-availability attack as it is the easiest to perform form the attacker's viewpoint, recall Section 2.6.2. One can also argue that it is the most difficult attack to detect as the attacker does not force the legitimate

client off the network[1]. It should also be noted that an algorithm that is able to detect the wait-for-availability attack will also be able to detect the session hijacking attack. This is due to the fact that the algorithm only take into consideration that a new STA, with a new fingerprint is active with the same MAC address. Not how the old (legitimate) STA is treated.

3.2.3 Man in the Middle

The MITM attack is a more complex attack that requires one to setup a fake AP. This attack no longer requires dedicated additional hardware as an AP can be setup in software — for instance on a Windows computer [26].

Given the limitations of our attacker as defined in the threat model, software MITM is still a viable method to gain access the network and the Internet masquerading as someone else. At the same time this attack is more in the class of phishing[2] and does not share many similarities with the wait-for-availability attack, it is therefore not considered part of the scope in this thesis.

3.3 Fingerprinting as Detection Method

3.3.1 The Need for a New Detection Method

To the author's knowledge there currently does not exist any method to detect MAC spoofing attacks of the type where the attacker and victim is *not* connected simultaneously. The other variation of this attack is more thoroughly tested and several possible solutions exists [1]. These solutions are based on detecting abnormalities due to two STAs acting as one, and are not suitable to distinguish two STAs connected at different times.

Fingerprinting is a method that from a theoretical standpoint should fit this problem well as a good fingerprinting algorithm is able to identify subjects (STAs) and thereby enable a program to distinguish between two different STAs.

3.3.2 Strengths of MAC Layer Fingerprinting

Fingerprinting on the MAC layer is based on implementation differences as highlighted in the papers discussed in Section 1.5, Related Work. Many of the properties mentioned are not easily modifiable as they are hardcoded in the drivers of the NIC.

[1]Forcing the client off the network requires sending of deauthenticate or channel switch frames. These frames can easily be observed by an IDS, ergo it can easily be detected.

[2]Phishing is a way of attempting to acquire sensitive information by masquerading as a trustworthy entity in an electronic communication [27].

Using properties that are hard to spoof is a very important part in creating a fingerprinting algorithm that is reliable.

The MAC layer properties are unencrypted, even in encrypted networks. These properties include frame type, power management information and other properties form the MAC header. This fact makes fingerprinting on the MAC layer more versatile than using properties from higher levels in the OSI model that might be encrypted. Another strength is that these properties are available through passive observation, not requiring any interaction with the STAs from the IDS.

3.4 Potential Fingerprinting Properties

3.4.1 Null Data Behavior

In the paper *On Security Vulnerabilities of Null Data Frames in IEEE 802.11 based WLANs* [4] the authors create seven rules to identify different behavior regarding the usage of Null Data frames. These rules can be seen below.

Rule 1 Some NICs use Null Data frames for power management (Y) while others use PS-Poll control frames (N)

Rule 2 Some NICs send Null Data frames once per 10 seconds when there is no data communication to keep association alive (Y), while others do not (N)

Rule 3 Some NICs send Null Data frames before sending Probe Request frames during active channel scanning (Y), while others do not (N)

Rule 4 When switching mode from PS mode to AM, some NICs send a Null Data frame (*PwrMgmt* set to 0) before sending the first data frame (Y), while others send the data frame directly (N)

Rule 5 Some NICs can still receive data frames for a short period of time after sending a Null Data frame (*PwrMgmt* set to 1) (Y), while others switch to the *Doze* state immediately after sending such a Null Data frame (N)

Rule 6 Some NICs send Null Data frames periodically (Y), while others send it aperiodically depending on the availability of data (N)

Rule 7 Some NICs send a Null Data frame (*PwrMgmt* set to 1) and switch to the *Doze* state after sending out all its data frames to the AP even if there are more buffered data frames at the AP (Y), while others do so only when there are no frames in both directions (N)

These rules motivated this work and form the basis of the algorithm developed in this thesis. A detailed explanation of how five of these rules are tested for will be presented in Section 3.5. Two of the rules are not applicable in our setting, these rules are Rule 5 and Rule 7 as they cannot be determined by merely observing. Both rules require access to some internal information from the STA in order to determine (Y) or (N). Rule 5 requires access to the internal power state while Rule 7 requires knowledge about whether the STA have sent all its packets or if it just entered PS mode periodically. These rules will therefore not be included in the fingerprinting algorithm developed.

3.4.2 Duration Calculation

Recall that the MAC header have a field called `Duration/ID`. In data frames this field contains a value that indicate the duration of the current transmission in microseconds [6]. The standard specifies that this calculation is dependent on the PHY in use as well as the transmission rate, still an algorithm for determining the duration is not specified. This point is stated in the paper by Gopinath et al. [3].

Not having a specified algorithm has led to several different implementations, the result is that identical frames sent over the same PHY with the same rate get different values in the `Duration/ID` field. Not many data frames are identical, but one particular type is — the Null Data frame. The value of the `Duration/ID` field in Null Data frames is therefore suited as a fingerprinting property.

3.4.3 Implicit Identifiers

As mentioned in related works the authors of [3] describes several ways to fingerprint a 802.11 device. The authors mention vendor specific extensions as a source for fingerprinting. Such extensions are included in the Association Request frame. This frame contains a lot of information about the STA, and it turns out that most STAs sends Association Request frames that differ from each other in several ways, not only in regards to vendor specific extensions. Fields or values whose main function is to inform the network about specific traits of the STA will be called implicit identifiers in this thesis. Examples of such fields in the Association Request frame are `Supported rates`, `Listen interval` and `Vendor Specific` amongst others.

3.4.4 QoS Usage

In [1] Idland points out that some STAs uses QoS Null Data frames while others do not. Even among those who do use QoS there are differences in which TID class they use. These properties will be used in the fingerprinting algorithm.

3.4.5 Other Properties

Some other potential fingerprinting properties have been mentioned in the related literature, this included differences in the random backoff algorithm (not random in some cases), reassociation latency, unique rate switching behavior and honor of the duration field as mentioned in [3]. None of these properties are used further in this thesis as they seem inferior in terms of being deterministic. Being able to determine whether a rule is upheld or not with little available traffic, and within a short timeframe is an important feature of the fingerprinting algorithm in our case.

In the paper by Franklin et al. [9] they present a fingerprinting technique based on statistical analysis of the inter-frame timing of transmitted Probe Request frames. The authors had good results with the inter-frame timing method and concluded that the majority of wireless drivers do have a distinct fingerprint. As this method is already implemented and tested it is not included in the algorithm developed in this thesis. It could however be included at a later stage to increase the number of different tests and potentially increase the success rate.

3.5 Developing the Fingerprinting Algorithm

3.5.1 Fingerprinting on the Fly

The first issue with developing the fingerprinting algorithm, hereafter referred to as the FPA, was to figure out how the fingerprints were to be used.

The traditional method is to have a database with known fingerprints for known subjects with specific settings. This would translate into generating a database for a set of STAs with different settings (QoS and probe settings can alter the fingerprint) and then identify STAs on the network as one type or another. This would in theory be good, but it would require a very large pre-computed database in order to be usable in a random network with arbitrary set of STAs. Another downside with this approach is that it in many cases would require a full fingerprint[3] in order to determine the exact match in the database.

The goal with the FPA is to detect a specific type of MAC spoofing attack, in other words being able to distinguish between two different STAs by observing them. This is more relaxed goal than being able to identify a given STA in terms of NIC, drivers, operating system (OS) and so on. It became evident that due to this goal another method could be used in order to detect unlikely changes in the behavior. This method does not share the two downsides with the traditional method, we call this method fingerprinting on the fly.

[3] A full fingerprint denotes a fingerprint created with every available property and not a subset of them.

Fingerprinting on the fly starts off as regular fingerprinting. As the necessary information is observed certain traits or properties are determined one by one. The big difference compared to the traditional method is that we do not wish to compare our fingerprint to pre-computed fingerprints in a database. There is also no need for a full fingerprint as a partial fingerprint, with only a subset of the properties determined, should be sufficient in many situations. The theory is that a change of STAs would with high probability result in a change in one or more of the properties already determined.

3.5.2 Expressing the Tests in Pseudocode

The next step in the development of the FPA is to create a procedure for determining each of the properties that should make up the fingerprint. The procedure or algorithm for determining a property is called a test.

The remainder of this section will be devoted to explaining and describing these tests. The tests are presented under the assumption that we are only looking at frames originating from one single MAC address (supposedly one STA). This is done in order to simplify the code and remove logic needed to identify the STA by MAC source address or AID, as well as keeping track of different STAs' fingerprints. The actual implementation includes these features and the entire Perl source code is available in Appendix C.1.

3.5.3 Test 1, PS-Poll

Test 1 is based on Rule 1 from [4], described in Section 3.4.1. Rule 1 states that some NICs uses Null Data frames and others uses PS-poll frames for power management. That is somewhat imprecise in relation to what was observed in the experiments done in this thesis. All STAs in the experiment used Null Data for changing power mode, but one STA used PS-Poll frames when the AP had buffered frames for it.

The test is therefore whether a STA use PS-Poll or not. If a STA use PS-Poll the FPA should observe a PS-Poll frame from the STA after the AP has announced that it has buffered frames for it. This PS-Poll frame should be observed within the `Listen interval` of that STA. The pseudocode for Test 1 can be seen in Algorithm 1 and an explanation will follow.

If the frame is a Beacon frame and the bit in the TIM corresponding to the STA in question is set the `beacon_count` is incremented. If the `beacon_count` exceeds the `Listen interval` then the algorithm concludes that PS-Poll is not in use and returns suspect attack if it previously was in use. When a PS-Poll frame is observed the `beacon_count` is reset and the algorithm naturally concludes that PS-Poll is in use, if that was not the case before the algorithm will return suspect attack.

Algorithm 1 Test 1, PS-Poll

Input: $frame$

if $frame$ == Beacon frame **and** bit in TIM is set **then**
  $beacon\_count$++
  if $beacon\_count > listen\_interval$ **then**
    use_PSPoll = **false**
    if use_PSPoll was **true then**
      return suspect attack

if $frame$ == PS-Poll frame **then**
  $beacon\_count = 0$
  use_PSPoll = **true**
  if use_PSPoll was **false then**
    return suspect attack

3.5.4 Test 2, Keep Alive

Test 2 is a direct implementation of Rule 2. The code found in Algorithm 2 tests whether a STA sends a Null Data frame if it has been idle for 10 seconds in order to keep the session alive. The timestamps are in milliseconds and thus testing for exactly 10 seconds would yield very few hits, therefore the test is implemented with a buffer, currently set to 0.15 seconds.

Algorithm 2 Test 2, Keep Alive

Input: $frame$

$time\_delta$ = timestamp previous frame - timestamp current frame
buffer is 0.15 sec
$within\_buffer = 9.85 < time\_delta < 10.15$

if $within\_buffer$ **and** $frame$ == Null Data **then**
  use_keep_alive = **true**
  if use_keep_alive was **false then**
    return suspect attack

else if $time\_delta > 10.15$ **then**
  use_keep_alive = **false**
  if use_keep_alive was **true then**
    return suspect attack

A step by step explanation of the algorithm will now be given. First the `time_delta` is calculated from the timestamp of the previous frame and the current frame. Then the algorithm checks if the `time_delta` is within the range of

10 seconds ± the buffer of 0.15 second and in addition if the frame is a Null Data frame. If both of these conditions is true the algorithm concludes that keep alive is in use and returns suspect attack if keep alive was not in use prior to this frame.

If the conditions in the first if-statement were not met a new if-statement checks if the `time_delta` is larger than 10.15 seconds (10 + buffer). If that is the case this is interpreted as the STA not using keep alive, and thus the algorithm concludes so and returns suspect attack if this was not the case previously.

3.5.5 Test 3, Null before Probe

Test 3 is also based on the rules given in [4], specifically Rule 3. The rationale behind this test is that some STAs sends a Null Data frame and goes into PS mode before starting the channel scanning with Probe Request frames while other STAs do not do this.

Algorithm 3 Test 3, Null before Probe

Input: $frame$

$set\_size = 5$
$min\_limit = 0.80$

if $frame ==$ Probe Request **then**

  if previous frame was Null Data **then**
    $using\_count{+}{+}$
    $total\_count{+}{+}$
  else if previous frame was **not** Probe Request **then**
    $total\_count{+}{+}$

if $total\_count == set\_size$ **then**
  $use\_null\_before\_probe = using\_count > set\_size \times min\_limit$
  if $use\_null\_before\_probe$ changed value **then**
    return suspect attack
  $using\_count = 0$
  $total\_count = 0$

The algorithm for this particular test is pretty uncomplicated and can be seen in Algorithm 3. First a set size and a minimum limit is defined. The set size is the number of probe request bursts that will be observed before any conclusion is made. The minimum limit is a number $\in [0, 1]$ representing the percentage of probe request bursts where the STA first sends a Null Data frame required in order for the algorithm to conclude that null before probe is in use.

If the current frame is a Probe Request frame the algorithm continues. If the

previous frame was a Null Data frame null before probe is in use and the algorithm increment the `using_count` as well as the `total_count`. On the other hand, if the previous frame was not a Null Data frame or a Probe Request frame then null before probe is not in use and the algorithm only increments the `total_count`.

The last part of the algorithm checks if the percentage of times null before probe was in use is over the minimum threshold in order to conclude if it in fact is in use. Suspect attack is returned whenever the current conclusion differs from the previous conclusion.

3.5.6 Test 4, Mode changing Null Data

Test 4 is almost a direct implementation of Rule 4, the only difference is regarding the `Listen interval` check as explained below. Test 4 checks if the STA, when it have data frames to send, uses a Null Data frame to change mode or if it directly sends a regular data frame when changing power mode. During the experiments it was noted that some STAs always use mode changing Null Data except when they have been in PS mode for a duration equal to their own `Listen interval`, this is therefore included in the test.

The pseudocode can be seen in Algorithm 4 and an explanation will now follow. First a set size and a minimum limit is defined. The set size is the number of power mode changes that will be observed before any conclusion is made. The minimum limit is a number $\in [0, 1]$ representing the percentage of power mode changes that must be done by using Null Data frames in order for the algorithm to conclude that mode changing Null Data is in use.

The first if-statement checks whether the STA has been in PS mode longer than its `Listen interval`, and if so changes the recorded power mode of the STA to AM.

The next if-statement checks if the `chk_nxt_pkt` variable is set. The first time the algorithm is executed this is not the case and the algorithm continues to check if the STA is in PS mode. If the STA is in PS mode (before the current frame) the algorithm checks if the current frame is a Null Data frame that changes the power mode to AM. If this is the case, the `chk_nxt_pkt` variable is set. If it is not a Null Data frame with `pwr_mgt` bit $== 0$, but a regular data frame with `pwr_mgt` bit $== 0$ the algorithm interpret this as the STA is not using null before probe and therefore increments the `total_count` without incrementing the `using_count`.

The reason for having the `chk_nxt_pkt` varialbe is that we are only interested in the power mode changes made when the next packet is a data packet. So, when the next packet is processed in the algorithm the `chk_nxt_pkt` variable is set and the algorithm checks if the current packet is a regular data packet. If this is the case

Algorithm 4 Test 4, Mode changing Null Data

Input: *frame*

```
set_size = 30
min_limit = 0.9

if frame == Beacon frame then
  if number of Beacons since data > listen_interval then
    pwr_mode = AM

if chk_nxt_pkt == true then
  if frame type is data and fame ≠ Null Data then
    using_count++
    total_count++
  chk_nxt_pkt = false
else
  if pwr_mode == PS then
    if frame == Null Data and pwr_mgt bit == 0 then
      chk_nxt_pkt = true
    else if frame is DATA and pwr_mgt bit == 0 then
      total_count++

pwr_mode = pwr_mgt bit (1 = PS, 0 = AM)

if total_count == set_size then
  use_mode_chng_null = using_count > set_size × min_limit
  if use_mode_chng_null changed value then
    return suspect attack
  using_count = 0
  total_count = 0
```

the `using_count` is incremented as well as the `total_count`.

The last part of the algorithm works the same way as in Algorithm 3 and returns suspect attack if the overall conclusion has changed since last time.

3.5.7 Test 5, Fixed Interval

Test 5 is based on Rule 6, but instead of looking at the interval between any two Null Data frames this test checks the duration between Null Data frames with different values in the `Power Management` bit. In the case were the STA is using mode changing Null Data this interval would translate into the time the STA was in PS mode. The idea behind this test is that some STAs stays in PS mode for a fixed interval, regardless if they get new data to transmit.

Algorithm 5 Test 5, Fixed Interval

Input: *frame*

buffer = 0.2
set_size = 50
min_limit = 0.80

if *frame* == Null Data **then**
  time_delta = time elapsed since previous Null Data frame

  if *pwr_mgt_previous* == *PS* **and** *pwr_mgt_current* == *AM* **then**
    if *average* exists **then**
      if *time_delta* is within *average* ± 20% (buffer) **then**
        pair_ok_count++
      pair_total_count++
    else
      calculate *average* from first 10 pairs

  if *pair_total_count* == *set_size* **then**
    use_fixed_interval = (*pair_ok_count*/*set_size*) ≥ *min_limit*
    if *use_fixed_interval* changed value **then**
      return suspect attack
    pair_ok_count = 0
    pair_total_count = 0

The pseudocode for Test 5 can be seen in Algorithm 5. First three variables are defined, their use will become clear later. If the current frame is a Null Data frame the algorithm continues. If the `pwr_mgt` bit of the previous Null Data frame was 1 (PS) and the value in the current frame is 0 (AM) then we have a pair of Null Data frames that we want to examine further. If there exists an average, then the time delta between the two frames is compared to this average time ±20% (which

is the buffer defined in the beginning). If the time delta is in the specific range, then a counter for successful pairs (`pair_ok_count`) is incremented. Regardless of the comparison the `pair_total_count` is incremented.

The last part of the algorithm checks if the fraction of pairs, where the time delta is within the range of the average time plus/minus the buffer, of a set of pairs (equal to the `set_size`) is greater than the minimum limit defined. If this is the case, then the algorithm concludes that the STA do use a fixed interval and returns suspect attack if this was not the case previously and vice versa.

3.5.8 Test 6, Null Data Type

Test 6 is based on some of the observations done by Idland in his specialization project [1] regarding Null Data behavior. Recall from Section 2.3.4 that the 802.11 standard has two types of Null Data frames; the regular one and the QoS Null Data frame.

It turns out that in a network where QoS is enabled (as it is in WrT) some STAs uses the QoS Null Data while others do not use it. Amongst those who use the QoS version there is also differences in which QoS priority (TID class) they utilize. These implementation differences are what make Test 6 viable.

Algorithm 6 Test 6, Null Data Type

Input: *frame*

if *frame* == Null Data **then**
  categorize frame as QoS or regular
  if *frame* was QoS type **then**
    identify the priority class (TID)
  if *frame* differs from the previous Null Data frame **then**
    return suspect attack

The algorithm for Test 6 can be seen in Algorithm 6 and it is relative simple compared to the algorithms for the other tests. Nevertheless it identifies a viable fingerprinting property. The algorithm basically identifies the type of Null Data frame in use and its priority class (TID) if it was a QoS frame. The algorithm then returns suspect attack if the identified frame differs from the previous identified Null Data frame.

3.5.9 Test 7, Duration Calculation

Test 7 was motivated by the paper by Gopinath et al. [3]. Revisit Section 3.4.2 for a short discussion of duration calculation as a fingerprinting property and the

rationale behind this test. The theory is that Null Data frames have the exact same size, and thus should have the same duration when the data rate is the same. A difference in the `Duration/ID` field would indicate different implementations in the calculation algorithm that again indicates two different STAs.

Algorithm 7 Test 7, Duration Calculation

Input: *frame*

if *frame* == Null Data **then**
  get the duration value from the Duration/ID field
  get the data rate from the radiotap header

  compare duration for given data rate with previous value

  if differences in duration value for same data rate **then**
    return suspect attack

The algorithm for Test 7 is described in Algorithm 7. It basically works by recording the duration taken from the `Duration/ID` field in the Null Data frames for each data rate used. Recall that the data rate is available in the radiotap header. If an inconsistency is found the algorithm returns suspect attack.

3.5.10 Test 8, Association Request

Vendor specific extensions as a fingerprinting source is mentioned in the paper by Gopinath et al. [3]. This was the motivation to further investigate the Association Request frame looking for possible sources for fingerprinting. Several potential fields were identified and they are: `Duration/ID`, `Listen interval`, `Supported Rates`, `Extended Supported Rates`, `QoS Capability` and `Vendor Specific`.

Algorithm 8 Test 8, Association Request

Input: *frame*

if *frame* == Association Request **then**
  record implicit identifiers
  if Ass. Req. for same MAC address is recorded before **then**
    compare implicit identifiers from current and previous frame
    if inconsistencies in implicit identifiers **then**
      return suspect attack

The algorithm for Test 8 can be seen in Algorithm 8 and is very straight forward. When the input frame is an Association Request frame the relevant fields as

mentioned above, called implicit identifiers, are recorded. If an Association Request has been recorded for this MAC address before a check on each individual field is done and suspect attack is returned in case of any inconsistencies.

3.5.11 Creating a Compound Fingerprint

In order to avoid a high false negative rate when using fingerprinting it is important to rely on several different properties and tests that can flag suspicious behavior. The fingerprint created in the experiments is a composition of properties determined by the eight tests explained above. The first six tests result in one fingerprinting property each while to two latter results in several properties. Each of the properties from the two latter tests can be used to detect suspicious behavior in their own right. See Table 3.1 for an exhaustive list of fingerprinting properties and their possible values that make up the compound fingerprint. For a more in-depth explanation of the possible values and their usage consult the 802.11 standard [6].

Fingerprinting Property	Possible Values
PS-Poll	True / False
Keep Alive	True / False
Null before Probe	True / False
Mode changing Null Data	True / False
Fixed Interval	True / False
Null Data Type	Regular/QoS including TID
Duration Calculation	Pairs of (rate, duration) for each rate
Association Request Duration	[0...32767]
Listen Interval	[0...256]
Supported Rates	Set of up to eight integers $\in$ [2...127]
Extended Supported Rates	Set of up to 255 integers $\in$ [2...127]
QoS Capability	Present / Not Present
Vendor Specific	Type of vendor specific element

Table 3.1: The fingerprinting properties and their possible values

The final FPA is a combination of each of the eight tests and some additional logic to determine whether or not we are dealing with a MAC spoofing attack. The pseudocode can be seen in Algorithm 9. Each of the individual tests return suspect attack if they find anything suspicious according to the above described algorithms.

The logic to determine whether we are dealing with an attack based on the output from the tests has not been defined. As mentioned we want to do fingerprinting on the fly and not necessarily generate a complete fingerprint. How many properties that are needed in order to have sufficient low false positive and negative rates is uncertain. Some of the tests might prove to be sufficient alone, while others

Algorithm 9 The Fingerprinting Algorithm (FPA)

Input: $frame$

run Test 1, PS-Poll
run Test 2, Keep Alive
run Test 3, Null before Probe
run Test 4, Mode changing Null Data
run Test 5, Fixed Interval
run Test 6, Null Data Type
run Test 7, Duration Calculation
run Test 8, Association Request

evaluate outputs from tests 1-8

return attack / no attack based on evaluation

require at least one other test in combination in order to conclude attack with a high probability. The experiments and the following results will shed some light on these issues and they will be further discussed in Chapter 6.

Chapter 4

Performing the Experiments

This chapter will elaborate on how the experiments with the FPA were performed. This includes the setup for the implementation in Perl, how to capture traffic and save it for later analysis, an overview of the different STAs used, and the test scenarios.

4.1 Implementing the FPA in Perl

Some of the tests in the FPA, especially those regarding Null Data behavior, would be very tedious to perform by manual analysis. In order to be able to run several experiments with relative large packet captures these tests had to be automated. This was done by implementing them in Perl. Perl was chosen as it is suitable for packet handling, and because it was the language used in previously work on a very related problem by Idland [1].

The implementation of the FPA in Perl is called the fingerprinting program (FPP) in order to separate them. Test 1-5 and 8 is implemented in Perl and the source code is available in Appendix C.1. Test 6 and 7 are not implemented in Perl as it was feasible and convenient to perform the analysis required manually through the use of Wireshark [28].

The FPP can run in live mode, i.e. not requiring any packet capture input, and in offline mode on a capture file. Live mode was not used in the experiments as it was expected that changes were to be made in the algorithm as the development was done in iterations as explained in Section 1.3. It was therefore an obvious advantage to have the entire scenario saved to a capture file, ready to be executed again and again instead of actually performing the scenario all over again each time a change was made. The live mode is on the other hand is suitable in a real world situation, supporting fingerprinting on the fly.

4.2 Experimental Setup

Certain hardware, software and specific NIC configurations must be in place before the FPP will be able to run at all. The NIC configuration is crucial to ensure that all the relevant wireless traffic is captured, and that it is done in such a format that it is readable by the FPP. The setup presented in this section is exactly what was used in the experiments.

4.2.1 Equipment

The necessary equipment used can be seen in the list below.

- Dell desktop (Dimension 9150) with Ubuntu 10.10 — the Maverick Meerkat
- Wireless NIC, Atheros AR5001X+, for sniffing all wireless data
- Wired NIC for regular Internet access
- Cisco LAP from WrT, connected to their network

4.2.2 Drivers, Software and Packages

The NIC must be set in monitor mode and the radiotap header must be included in order for the FPP to work correctly. These configuration options might not be available through the standard drivers, and thus might require installation of other, more advanced drivers. In the setup used in the experiments the MadWifi tools and drivers [29] were used with good results.

In order to capture the traffic form the wireless network and save it for later analysis a packet sniffer had to be used, the program of choice was Wireshark.

Two of the tests were performed with manual analysis and this required a packet analyzer. An analyzing tool with powerful filtering and display options is recommended for this job, such a tool would also do well in case of any troubleshooting. The tool used throughout the work done in this thesis was the same as the one used for the packet sniffing, namely Wireshark.

Perl is required to run the FPP, and it was included in my Ubuntu distribution. Some additional packages are necessary in order to do the packet handling and these can be obtained through the synaptic packet manager. A list of software, drivers and packages is found below.

- MadWifi tools and drivers, version 0.9.4
- Wireshark, version 1.4.6
- Perl
  - Net::Packet
  - Net::Pcap

– Bit::Vector

4.2.3 Configuring the Wireless NIC

The wireless NIC that is used for packet capture has to be set in monitor mode, listening to a specific channel. It is also crucial to specify how the capture should be formatted and what is to be included in the capture. There are several different headers that can be included or excluded, but only one configuration will work with the FPP. The following commands will set up the wireless NIC correctly.

```
wlanconfig ath0 destroy                              # (1)
wlanconfig ath create wlandev wifi0 wlanmode mon     # (2)
iwconfig ath0 channel 6                              # (3)
ifconfig ath0 up                                     # (4)
echo '803' > /proc/sys/net/ath0/dev_type             # (5)

# (1) Destroy any previous interface
# (2) Create a new interface in monitor mode
# (3) Monitor a specific channel, here channel 6
# (4) Make the interface accessible to the IP layer
# (5) Include Radiotap headers
```

4.3 Capturing Packets

After the necessary setup is in place as explained in the above section the packet capture can begin. This is done by starting the capture in Wireshark on the interface `ath0`. If the capture is done in an environment with several other Wi-Fi networks one might consider to apply a filter, filtering out all packets not going to or originating from target LAP or the Nomadix gateway in order to reduce the size of the capture file. Recall the network topology from Figure 3.1 in Section 3.1.1.

When the relevant scenario is captured the file should be saved to disk by by choosing `save` in the file menu, it is then ready to be used as input to the FPP.

4.4 The Scenarios

Two main types of scenarios have been used in the experiments. The first type is called General Usage and it is a scenario simulating general usage and will be used to determine how unique the fingerprints are. The next type of scenario includes an actual attack. Both types have their place in the testing of the FPA.

4.4.1 Scenario 1, General Usage

Scenario 1, General Usage, is a scenario constructed to contain behavior needed to conclude each of the eight tests in the FPA. When creating this scenario two behavior patterns were considered important to include. First, the STA should generate traffic and have more than enough data to send for a period of time, resulting in little or no idle time. Second, the STA should have longer periods where there were no traffic to send. The rationale is that this is a realistic usage pattern as users often browse the web, start reading or watching something before they continue the browsing. A scenario including these patterns should also be able to elicit different power management and Null Data behavior.

The scenario was performed the exact same way for every STA. This was done in order to enable an accurate comparison of the results, as well as identify differences due to implementation variations, and not due to the way the STA was used. The exact scenario can be seen in Appendix B.1.

4.4.2 Scenario 2, Wait-for-Availability Attack

Scenario 2, Wait-for-availability Attack, is actually performed in two versions. In version 2a a STA, client A is operating for approximately eight minutes having some periods with a lot of traffic and other periods with no or less traffic similar to the behavior found in Scenario 1. Then, client A logs off and a client B spoofs his MAC address and performs the wait-for-availability attack. Client B continues to browse the web having some periods with little or no traffic as well as periods with a lot of traffic, this goes on for about eight minutes before the capture stops.

In this scenario two STAs were arbitrarily chosen from the pool of test STAs, one laptop (S-7) and one smartphone (S-5). Only this one combination of STAs were used as it turned out that the capture file from the scenario was more or less identical to the concatenation of the capture files form Scenario 1 for the two relevant STAs. There were in other words little or nothing to gain by repeating Scenario 2a for every possible combination of available STAs.

The second version, Scenario 2b, contains the same main scenario as in 2a, but in a much shorter time frame. The main difference is that now there are five additional STAs connected to the LAP generating traffic at the same time. In other words, there were a total of seven STAs connected to the LAP and generating traffic. As there were seven different STAs generating traffic the capture file grow very large very fast, and as a result the scenario was shortened down to a total of eight minutes (four minutes before the attack and four minutes after). This fact certainly makes it harder for the algorithm to determine the fingerprints as it will be a lot less data to analyze. The results will show if four minutes in an active network is enough, or

if longer observations as the ones in Scenario 2b and Scenario 1 are required.

4.5 The Sample STAs

In order to test the FPA the behavior of several different STAs had to be analyzed. In the experiments 10 different STAs were tested. They were not specifically chosen, but happened to be the ones available through WrT, friends and personal equipment of the author. As a result the set of STAs is relatively randomized and representative for what typically is connected to WrT any given day. None of the settings of the STAs were altered from how they were before being used in the experiments with one exception. One STA was used twice (S-4 and S-11), first in its regular setup and then with the use of Backtrack 5 [30]. This was done as Backtrack is regarded as a typical attacker OS, and it would therefore be interesting to see how the OS affects the behavior of the STA, and how its fingerprint compares to the other regular STAs.

The specification of each STA can be seen in Table 4.1. Seven of the STAs used in the experiments are laptops. The other STAs include two iPhones and HTC Hero which are all smartphones, and the iPod which is a smart music player. Even though there are only 10 different physical STAs they will be referred to as 11 due to the one STA with two different software setups.

No.	Name and Model	OS	NIC	Browser
S-1	Dell XPS	Windows 7	Intel WiFi 1000bgn	IE 8
S-2	Lenovo S10-3S	Windows 7	Broadcom 802.11n	Opera 11
S-3	Acer Aspire 5745G	Windows 7	Broadcom 802.11n	IE 8
S-4	Dell Inspiron 9400	Windows 7	Intel P/W 3945abg	Chrome 11
S-5	iPhone 1. gen.	iOS 3.0.1	Not available	Safari
S-6	iPhone 4. gen.	iOS 4.3	Not available	Safari
S-7	Dell Latitude D610	Windows 7	Intel P/W 2915abg	IE 8
S-8	Acer Aspire 5670	Win. XP SP3	Intel P/W 3945abg	Firefox 3.6
S-9	iPod touch 1. gen	iOS 3.1.3	Not available	Opera Mini 5
S-10	HTC Hero	Android 2.2.1	Not available	”Internet”
S-11	Dell Inspiron 9400	Backtrack 5	Intel P/W 3945abg	Firefox 4

Table 4.1: Overview of the STAs used in the experiments

4.6 Performing the Wait-for-Availability Attack

Scenario 2a and 2b requires that the wait-for-availability attack is performed. This is a basic MAC spoofing attack where the attacker waits until the legitimate clients

leaves the network before he tries to connect with the legitimate client's MAC address. Wireshark was used for the first part, i.e. it was used to find a potential victim, identify his MAC address and for observing when the traffic ceased. The setup required in order to use Wireshark can be found in Appendix A.1. The commands required to perform part two, the MAC spoofing, can be seen in Appendix A.2.

4.7 Obtaining the Results

After the scenarios were completed and the related packets were captured and saved to a .cap file it was time to apply the tests from the FPA in order to get some results that could be further analyzed. This part was done in two different ways. The more complex tests were solved by running the FPP as they have been implemented in Perl, while the simpler tests were completed by manually analyzing the packets.

4.7.1 Running the FPP

When running the FPP offline mode is used, as previously stated, and it is then important to set the dump-file variable to the capture file (.cap) that contains the captured frames one wants to analyze. The configuration used in the experiment can be seen in the source code in Appendix C.1.

The values of the variables can be seen in Table 4.2. For a recap these variables and their meaning consult the relevant test in Section 3.5.2. The program is then executed the same way as any Perl script: `./FPP.pl`.

Test	Variable	Value
Test 2, Keep Alive	buffer	0.15 sec
Test 3, Null before Probe	set_size	5
Test 3, Null before Probe	min_limit	80%
Test 4, Mode changing Null Data	set_size	30
Test 4, Mode changing Null Data	min_limit	90%
Test 5, Fixed Interval	buffer	20%
Test 5, Fixed Interval	set_size	50
Test 5, Fixed Interval	min_limit	80%

Table 4.2: Values of the variables in the experiments

4.7.2 Manually Analyzing Packets

Wireshark was used for the manual analysis as it has powerful filtering options. First, the capture was filtered on a given MAC address with this filter: `wlan.da ==`

`STA MAC OR wlan.ra == STA MAC OR wlan.sa == STA MAC OR wlan.ta == STA MAC OR (wlan.da == ff:ff:ff:ff:ff:ff AND wlan.sa == LAP MAC)`. That ensured that only packets going to or from that address as well as Beacon frames and other frames sent to everyone from the LAP was included. Then other more specific filters were used as for instance this one: `wlan.fc.type_subtype == 0x2c AND wlan.fc.type_subtype == 0x24` that filters out all QoS Null Data and regular Null Data frames.

Chapter 5

Results

This chapter will present the results obtained by executing the tests in the FPA either by use of the FPP, or by manual analysis of the packet captures. The capture files containing the packets used were obtained by performing the different scenarios as described in Section 4.4.

5.1 Scenario 1, General Usage

The results from each of the eight tests performed on the capture files from Scenario 1 from each of the 11 different STAs is presented in Table 5.1 and Table 5.2. The first column in each table presents the different fingerprinting properties, each of the other columns represents one STA. Each vertical column below a STA is in fact the fingerprint of that particualar STA.

Fingerprinting Property	S-1	S-2	S-3	S-4	S-5	S-6
PS-Poll	F	F	F	F	F	F
Keep Alive	**T**	F	F	—	F	F
Null before Probe	**T**	—	**T**	**T**	**T**	—
Mode changing Null	F	F	F	**T**	F	F
Fixed Interval	F	F	F	F	F	F
Null Data Type	QoS 0	Regular	Regular	QoS 0	QoS 7	Regular
Duration Calculation	314	44	44	314	258	44
Ass. Req. Duration	60	314	213	314	314	314
Listen Interval	10	10	10	10	10	10
Supported Rates	s1	s2	s3	s3	s3	s2
Ext. Sup. Rates	e1	e2	e1	e1	e1	e2
QoS Capability	F	F	F	F	F	F
Vendor Specific	v1-v4	v1,v5	v1-v4	v1	v1	v1,v5

Table 5.1: Results from Scenario 1 on tests 1-8 for S-1 to S-6. Detailed results from Duration Calculation can be seen in Table 5.3

Fingerprinting Property	S-7	S-8	S-9	S-10	S-11
PS-Poll	F	F	F	**T**	F
Keep Alive	F	F	F	F	F
Null before Probe	**T**	**T**	F	—	F
Mode changing Null	**T**	**T**	F	F	—
Fixed Interval	**T**	F	—	F	—
Null Data Type	Regular	QoS 0	QoS 7	Regular	Regular
Duration Calculation	44,314	44,314	258	44,213,223,258	44
Ass. Req. Duration	314	213	314	258	314
Listen Interval	10	10	10	3	5
Supported Rates	s3	s1	s3	s4	s1
Ext. Sup. Rates	e1	e1	e1	e3	e1
QoS Capability	F	F	F	**T**	F
Vendor Specific	v1	v1-v4	v1	v1	v1

Table 5.2: Results from Scenario 1 on tests 1-8 for S-7 to S-11. Detailed results from duration calculation can be seen in Table 5.3

The first five fingerprinting properties have one of two possible values, T representing *True* and F representing *False*. These values are determined as described in Algorithms 1-5 with the values as noted in Table 4.2.

The Null Data Type, Ass. Req. Duration and Listen Interval should be self explanatory, the Duration Calculation on the other hand is more cryptic and the detailed results of this test can be seen in Table 5.3. The values each STA have for the Supported Rates and Ext. Sup. Rates property is a category. These categories represent different sets of data rates as shown in Table 5.7 in Section 5.3. The QoS Capability is either present or not present, here denoted by T or F respectively. The values in the Vendor Specific property refer to a short name for the vendor specific information element, the explanation of these short names can be seen in Table 5.8 Section 5.3.

In Table 5.1 and Table 5.2 some STAs have the value "—" instead of T or F in property 2-5. These properties are determined by tests 2-5, and these tests do all require a minimum number of observations before a conclusion can be made. The dash is a symbol for undetermined, and is used when the required minimum number is not met.

Figure 5.1 depicts the Hamming Distance[1] between the fingerprints shown in Table 5.1 and Table 5.2. When calculating the Hamming distance the last six fingerprinting properties count as one property as they all depend on a single frame, namely the Association Request frame, that could be dropped or faked. In other words, each of the eight tests count as one property, and thus the maximum distance

[1]The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different [31]

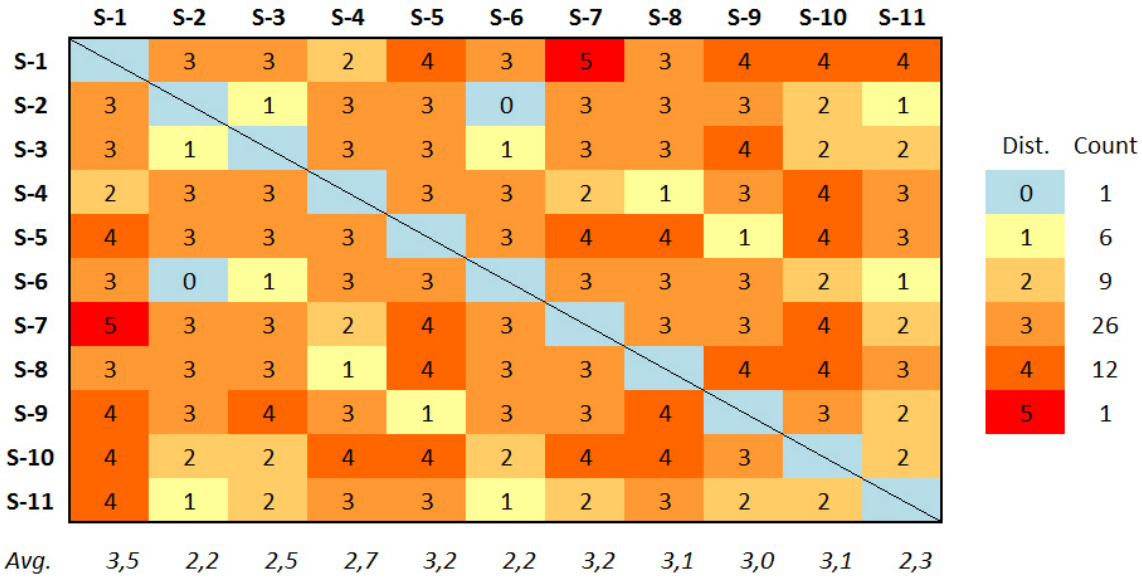

Figure 5.1: Hamming Distance for the fingerprints in Scenario 1

is eight. The average distance for each STA to every other STA ranges from 2.2 to 3.5 i.e. some STAs display a more unique behavior than others. The average of the averages is 2.82.

Data Rate	S-1	S-2	S-3	S-4	S-5	S-6	S-7	S-8	S-9	S-10	S-11
1	314	—	—	314	258	—	314	314	—	—	—
2	—	—	—	—	258	—	—	—	—	258	—
5.5	—	—	—	—	—	—	—	—	—	223	—
6	—	—	—	—	258	—	—	—	—	—	—
9	—	—	—	—	—	—	—	—	—	—	—
11	—	—	—	—	—	—	—	—	—	213	—
12	—	—	—	—	258	—	—	—	—	—	—
18	—	—	—	—	258	—	—	—	—	—	—
24	—	—	—	—	258	—	—	44	—	44	—
36	—	44	44	—	258	44	—	—	258	—	—
48	—	44	—	—	—	44	44	44	—	—	—
54	—	44	44	—	—	44	44	—	—	—	44
№ of frames	3008	406	770	432	734	1802	1698	307	51	113	1

Table 5.3: Detailed results from Duration Calculation in Scenario 1: Observed duration of Null Data frames for each possible data rate

Table 5.3 shows the detailed result from Test 7, the first column contains every data rate in 802.11a/b/g. Every STA used in the experiments supports each of these rates. After the first column, each STA have its own column with the observed duration for a Null Data frame for the given data rate. The dashes indicate that no Null Data frame was observed on that particular data rate. The last row shows the total number of Null Data frames observed for a given STA during Scenario 1.

5.2 Scenario 2, Wait-for-Availability Attack

In both Scenario 2a and 2b the attacker's Authentication Request frame was observed, making it easy to identify exactly when the attack occured. The fingerprints from the two scenarios can be seen in Table 5.4 and Table 5.5.

Fingerprinting Property	S-5	S-7
PS-Poll	F	F
Keep Alive	F	F
Null before Probe	**T**	**T**
Mode changing Null	F	**T**
Fixed Interval	F	F
Null Data Type	QoS 7	Regular
Duration Calculation	258	44,223,258,314
Ass. Req. Duration	314	314
Listen Interval	10	10
Supported Rates	s3	s3
Ext. Sup. Rates	e1	e1
QoS Capability	F	F
Vendor Specific	v1	v1

Table 5.4: Results from Scenario 2a on tests 1-8 for S-5 and S-7. Detailed results from Duration Calculation can be seen in Table 5.6

The fingerprints of S-5 and S-7 are identical to the fingerprints found in Table 5.1 and Table 5.2 with the exception of Fixed Interval for S-7. There were now observed Null Data frames on four additional data rates for S-7 compared to what was observed in Scenario 1. These particular results will be commented on in Chapter 6, Discussion and Analysis.

Table 5.6 shows the detailed result from Test 7, Duration Calculation, for Scenario 2a and 2b. This data is presented the exact same way as in Table 5.6. The durations marked with the † symbol are durations that were not observed in Scenario 1. The $*$ symbol on the other hand denote durations that were observed in Scenario 1, but not in Scenario 2a or 2b. Notice that S-7 only sent a total of 10 Null

Fingerprinting Property	S-5	S-7
PS-Poll	F	F
Keep Alive	F	F
Null before Probe	—	**T**
Mode changing Null	—	—
Fixed Interval	—	—
Null Data Type	QoS 7	Regular
Duration Calculation	258	314
Ass. Req. Duration	314	314
Listen Interval	10	10
Supported Rates	s3	s3
Ext. Sup. Rates	e1	e1
QoS Capability	F	F
Vendor Specific	v1	v1

Table 5.5: Results from Scenario 2b on tests 1-8 for S-5 and S-7. Detailed results from Duration Calculation can be seen in Table 5.6

Data Rate	S-5 (2a)	S-7 (2a)	S-5 (2b)	S-7 (2b)
1	258	314	258	314
2	258	258†	258	—
5.5	—	223†	—	—
6	258	—	258	—
9	—	—	—	—
11	—	—	—	—
12	258	—	258	—
18	258	—	—∗	—
24	258	44†	258	—
36	258	44†	258	—
48	—	44	—	—∗
54	—	44	—	—∗
№ of frames	439	562	264	10

Table 5.6: Detailed results from Duration Calculation in Scenario 2: Observed duration of Null Data frames for each possible data rate

Data frames in Scenario 2b compared to 562 in Scenario 2a. Recall that Scenario 2b lasted around four minutes, compared to eight minutes for 2a.

5.3 Categories and Short Names

Table 5.7 presents the categories for supported rates and extended supported rates and Table 5.8 shows the short names for the vendor specific information elements. Both categories and short names were used in the result tables in the previous sections.

Category	Data Rates	Note
s1	1, 2, 5.5, 11, 6, 9, 12, 18	802.11b rates first
s2	1, 2, 5.5, 11, 18, 24, 36, 54	
s3	1, 2, 5.5, 6, 9, 11, 12, 18	As s3, but ordered from low to high
s4	1, 2, 5.5, 11	Only basic and 802.11b rates
e1	24, 36, 48, 54	Four highest 802.11a/g rates
e2	6, 9, 12, 48	Compliment of s2
e3	6, 9, 12, 18, 24, 36, 48, 54	All 802.11a/g rates

Table 5.7: Categories of supported and extended supported rates

Short Name	Vendor	Details
v1	Microsoft	WME
v2	Aironet	Aironet CCX version = 4
v3	Aironet	Aironet Unknown
v4	Cisco	Cisco CCX1 CKIP + Device Name
v5	Broadcom	

Table 5.8: Short names for vendor specific information elements

Chapter 6

Discussion and Analysis

This chapter will discuss the results obtained in terms of the uniqueness of the fingerprints, the validity in a general perspective and how the FPA would do as a detection method in a real world IDS.

6.1 Uniqueness of the Fingerprints

As mentioned before it is not required that each STA has a unique fingerprint. What is, important is that the probability that two arbitrarily chosen STAs have different fingerprints is high. Figure 5.1 shows in great detail how different each STAs' fingerprint is. Except from when you compare a STA to itself there is only one single occurrence where the Hamming distance is zero, i.e. the fingerprints are identical. This is true for the couple consisting of S-2 (Lenovo laptop) and S-6 (iPhone 4G). Both of these STAs' fingerprints are undetermined in Test 3, Null before Probe, so it is possible that with more data they would end up being different as well.

Comparing the fingerprints, 29 of 55 (52.7%) out of the possible couples (not counting a STA with itself) have a Hamming distance of three or higher, and the average distance among all STAs is 2.82. These numbers are relativly high, and they indicate that the STAs do in fact have a unique observable behavior.

A high Hamming distance on a full fingerprint allows for a non-zero Hamming distance on several possible partial fingerprints which clearly is what we would like in a good IDS.

6.2 The Validity of the Results

The experiments were done in the actual WrT network with different STAs ranging from large and small laptops to smartphones and music players. The scenario was

performed by a human user, and not a script or the like, and it was done with the standard settings in use. With that in mind the result should be very close or identical to what one could obtain by actually monitoring an active part of the WrT network, i.e. the results seem valid from a general perspective.

6.3 The FPA as Detection Method

This section will present a discussion on how the FPA would perform as the main detection method in a IDS targeting the wait-for-availability or session hijacking attack.

6.3.1 Spoofing a Fingerprint

In order to be a good detection method the properties that the algorithm relies on should not be easily spoofable. Every property used in the FPA originates from transmitted bits, in other words it should theoretically be possible to alter them. From the results we know that S-4 and S-11 is in fact the same physical STA, but with Windows 7 and Backtrack 5 respectively. The Hamming distance of their fingerprints is three. This is due to the Backtrack version not using power management at all, and only sent out one single Null Data frame. The Association Request frames even have different parameters.

The case with S-4 and S-11 indicates that everything is in fact dependent on drivers and settings, but how easy is it to alter a STA's behaviour to match a certain fingerprint? Some drivers have settings for power management that should alter the behavior in the first five tests. Changing the duration calculation and Association Request frame parameters on the other hand is probably not that easy as they are typically hard coded in the drivers. Turning everything off like Backtrack 5 does is not very helpful, as no other STA does that. An easy, but potentially very time requiring approach is to wait for a STA with an identical fingerprint before performing the attack.

How much a fingerprint changes by altering the power management settings, and if it is possible to change it to match a specific STA should be tested in order to obtain a better understanding of exactly how hard it is to fool a system based on the FPA. This work is, as noted in Section 1.2, out of scope for this thesis due to time limitations as the development and testing of the FPA was considered more important. It should however be done before the FPA is utilized in a real setting, and it is therefore naturally recommended for future work.

6.3.2 Fingerprinting on the Fly

Recall that we want to do fingerprinting on the fly and be able to make decisions on attack / no attack based on partial fingerprints. Scenario 1 lasted for about eight to ten minutes, that was enough to create more or less complete fingerprints for all STAs involved.

The only test that require more time across all STAs was Test 7, Duration Calculation. With more time, and more STAs connected to the LAP we might see an increase in the number of observed data rates used. The results from Scenario 2a included Null Data frames observed on four new data rates for one of the two STAs (S-7). This is a clear indication that with more time and data more duration/data rate pairs will be observed.

The results from Scenario 2b showed that a significant decrease in the time spent observing, going from eight minutes (Scenario 2a) to four minutes, had a large impact on the fingerprint. It also showed that one of the two STAs sent distinctively less Null Data frames when the network was more congested. S-7 went from transmitting 562 frames in 2a to only 10 frames in 2b, this was not the case for the other STA as we can see from Table 5.6. Nevertheless the fingerprints created from Scenario 2b differed on two properties, and that should be enough to conclude attack.

6.4 Comparison with a Commercial Wireless IDS

A comparison of a commercial wireless IDS with the FPA when dealing with the hijacking or wait-for-availability attack would be very interesting in order to determine if the developed algorithm in fact increased the probability to detect such attacks. Practical comparisons were not feasible due to time and resource constraints, but a theoretical comparison could be performed.

Several wireless equipment providers as Cisco and Aruba Networks state that they provide IDSs without giving any detailed information. Different vendors were contacted through Wireless Trondheim as they are in fact seeking a good IDS solution. A reply with some detailed information was given by company called fdXtended regarding their product HSMX which is an Internet access platform implemented with a wireless IDS. It is a system that provides captive portal functionality and IDS features for open 802.11 networks [32]. This system is a direct competitor to WrT's Nomadix gateway. The following quote is taken from the HSMX datasheet under intrusion detection:

> Spoofing detection: MAC address spoofing and fixed IP spoofing detection. Hijacking prevention: A heartbeat between the console of a user

> and the HSMX ensures nobody can spoof that MAC address and gain access to the Internet without authentication [32].

Spoofing detection and hijacking prevention sounds very promising. In order to determine which techniques that were used here fdXtended was contacted through WrT. The heartbeat used for hijacking prevention is in fact the live SSL window technique mentioned in 1.5.2 and require two active browser windows at the same time. fdXtended claims that it does work on the iPhone 4 and iPad, but they had not tested it on other smart clients. The spoofing detection was based on not accepting more than one single IP address for a given MAC address. When presenting them for the wait-for-availability attack this was the response:

> Indeed in this case user B will get the IP of user A and there is no way to tell the difference between the two users, but obviously simultaneous [access] will not work.

The HSMX did in other words not have any possibility to detect the wait-for-availability attack and thus the FPA would substantially increase the probability to detect such attacks compared to this commercial IDS.

Chapter 7

Future Work

This chapter will present ideas for future work. The chapter will start by presenting additional tests that might augment the uniqueness of the fingerprint, then it will continue with a recommendation for determining the difficulty of spoofing the fingerprinting properties. Finally some ideas for how to improve the threshold tests are presented.

7.1 Additional Tests

During the experiments and analysis some behavior patterns were noted that might be used as bases for new tests. These tests would then add more fingerprinting properties to the fingerprint, and as a result probably increase the uniqueness of the fingerprints.

Use Power Management The STA with Backtrack 5, S-11, did not use power management at all. That is, none of its frames had the `pwr_mgt` bit set to 1. Every other STA used power management.

Probe Request Interval During the experiments in this thesis it was discovered that some STAs sent out Probe Request frames with a regular interval, while others did not. This relates to the time between sets of Probe Request frames, and not single frames.

A test on inter-frame timing of transmitted Probe Request frames has already been implemented by Franklin et al. [9] with good results and could be added to the FPA.

Prioritized Data Rate From the data in Table 5.3 it seems like some STAs use all available data rates when sending Null Data frames, while others only use one single data rate or only a few. For example during Scenario 1 S-5 sent 734 Null

Data frames divided on seven of its 12 supported data rates. S-1 on the other hand sent 3008 Null Data frames, and every single frame was sent on data rate 1.

Deauthentication versus Disassociation It turns out that some STAs use the Deauthentication frame while other use the Disassociation frame when the user disconnects from the network. A fingerprinting property based on this fact will not be determined before the client have logged off, in other words pretty late compared to the other tests. Nevertheless it could be the one test that made the difference, and it is mentioned here to show the variety of implementation differences.

Association Request The Association Request frame contains more parameters as `Supported Channels` and `Power Capability` that could be added to Test 8 in order to make the fingerprinting property based on the Association Request frame even more unique.

7.2 Difficulty of Spoofing the Properties

A comprehensive research into how difficult it is to alter each of the fingerprinting properties found in Table 3.1 as well as the new ones proposed in the section above would be very interesting. This should preferably be done on different STAs with different OSs as it is reasonably to believe that laptops with Linux is easier to alter than their Windows counterpart, or even smartphones.

7.3 Tuning the Logic

7.3.1 Conclude Attack

The final logic to determine if there was an attack or not needs to be specified. Some of the properties as Fixed Interval or Null before Probe is based on a threshold value and therefore includes some uncertainty. Other properties like PS-Poll, Duration Calculation and Null Data Type is constant for a given STA. Requiring differences in at least two of the more uncertain tests before concluding attack could be the optimal choice in terms of minimizing false positives and false negatives. On the other hand, a disparity in one of the tests not based on a threshold should be enough to conclude attack.

An investigation into which properties that can be used alone to conclude attack, and which that require others should be interesting for future work.

7.3.2 Threshold Tests

Test 2, 3, 4 and 5 is based on a threshold value. These values need some fine tuning in order to find the optimal value for minimizing false positives and false negatives. This would require larger data sets than what was used in this thesis.

An idea to further improve these tests is to introduce one additional state for each of the threshold properties, namely indifferent. Right now the threshold for *True* ranges from 80% to 90%. Everything below the threshold for a given test is by default *False*. In the experiments is was noted that some STAs had 0-10 of 50 in the Mode changing Null Data test, while other had 15-20 of 50 in the same test. Both these results was interpreted as *False*, but it could be an idea to introduce *indifferent* in order to separate them from each other and make the fingerprints more unique. Figure 7.1 shows the idea with two thresholds, one for *True* and one for *False*.

Figure 7.1: Slider depicting two thresholds, one for *True* and one for *False*

Chapter 8

Conclusion

The FPA has proved to create unique fingerprints. More precisely, fingerprints created on the basis of an 8-10 minute scenario exhibited an average Hamming distance of 2.82. In the case of severely reduced data the algorithm was still able to distinguish between STAs as evident from the results from Scenario 2b. No pre-computed database of fingerprints is required, reducing the need for storage and development of such a database, but equally important it allows for detection based on partial fingerprints.

The FPA works on the MAC layer and will therefore also work in networks where the upper layers are encrypted, as supposed to methods based on fingerprinting in the higher layers of the OSI model. Compared to a commercial IDS from fdXtended [32] the FPA is able to detect spoofing attacks of the type where the attacker and victim are not connected simultaneously.

Regarding the difficulty of faking a fingerprint it is known that identical STAs will have an identical fingerprint. Some fingerprinting properties are not easily altered as they are hard coded in the drivers, while others could potentially be easy to alter. Determining how difficult it is to alter a given fingerprint to match another one is proposed as future work. Nevertheless, the difficulty and skill required to avoid detection is raised considerably by implementing the FPA.

Considering the problem statement and task description from Section 1.2 it is evident that there now exists a way to detect the session hijacking attack, and the wait-for-availability attack in 802.11 networks, hence the following can be stated.

The FPA is capable of passively creating a fingerprint of wireless STAs without specialized equipment in realistic network conditions. Fingerprints from different STAs are unique with a high probability, even when there are little data available. In addition, the technique used is accurate, fast, and requires no pre-computed databases. The FPA used in combination with the IDS developed by Idland [1] is now able to detect all of the five different MAC spoofing attacks described in Section 2.6.2.

References

[1] Christer Idland. Detecting Identity Thefts in QoS Enabled Open 802.11 Wireless Networks. Technical report, Norwegian University of Science and Technology, 2010.

[2] Øystein Aas Pedersen. Detecting Wireless Identity Spoofs in Urban Settings, Based on Received Signal Strength Measurements. Master's thesis, Norwegian University of Science and Technology, 2010.

[3] K. N. Gopinath, Pravin Bhagwat, and K. Gopinath. An empirical analysis of heterogeneity in ieee 802.11 mac protocol implementations and its implications. In *Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, 2006.

[4] Wenjun Gu, Zhimin Yang, Can Que, Dong Xuan, and Weijia Jia. On Security Vulnerabilities of Null Data Frames in IEEE 802.11 based WLANs. In *Proceedings of The 28th International Conference on Distributed Computing Systems*, 2008.

[5] Rupinder Gill, Jason Smith, Mark Looi, and Andrew Clark. Passive Techniques for Detecting Session Hijacking Attacks in IEEE 802.11 Wireless Networks. In *AusCERT Asia Pacific Information Technology Security Conference Refereed R&D Stream*, pages 26–38, 2005.

[6] IEEE. IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Technical report, IEEE, 2007.

[7] Wireless Trondheim. Wireless Trondheim. `http://tradlosetrondheim.no/index.php?option=com_content&id=8`. Web-page retrieved 11th March 2011.

[8] Haidong Xia and José Brustoloni. Detecting and Blocking Unauthorized Access in Wi-Fi networks. In *Networking 2004*, pages 795–806. Springer-Verlag, 2004.

[9] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, 2006.

[10] Jon Edney and William A. Arbaugh. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Addison-Wesley Publishing Company, 2004.

[11] Rupinder Gill, Jason Smith, and Andrew Clark. Experiences in Passively Detecting Session Hijacking Attacks in IEEE 802.11 Networks. In *Proceedings of 4th Australasian Information Security Workshop (Network Security)*, 2006.

[12] Fanglu Guo and Tzi-cker Chiueh. Sequence Number-Based MAC Address Spoof Detection. In *International Symposium on Recent Advances in Intrusion Detection (RAID) 2005*, 2005.

[13] Eirik Holgernes. Detecting Identity Thefts in open 802.11e Enabled Wireless Networks. Master's thesis, Norwegian University of Science and Technology, 2010.

[14] Johannes Berg. Radiotap. `http://www.radiotap.org`. Web-page retrieved 15th April 2011.

[15] Johannes Berg. defined-fields/TSFT - radiotap.org. `http://www.radiotap.org/defined-fields/TSFT`. Web-page retrieved 15th April 2011.

[16] Bastian Könings, Florian Schaub, Frank Kargl, and Stefan Dietzel. Channel Switch and Quiet attack: New DoS attacks exploiting the 802.11 standard. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on Local Computer Networks (LCN)*, 2009.

[17] Nmap. Nmap Security Scanner. `http://nmap.org`. Web-page retrieved 5th April 2011.

[18] Nmap. OS Detection. Chapter 15, Nmap Reference Guide. `http://nmap.org/book/man-os-detection.html`. Web-page retrieved 5th April 2011.

[19] Peter Eckersley. How Unique Is Your Web Browser? In *Proceedings of the 10th international conference on Privacy enhancing technologies*, pages 1–18, 2010.

[20] Anton Chuvakin Cyrus Peikari. *Security Warrior*. O'Reilly Media, 2004.

[21] Michal Zalewski. [the new p0f]. `http://lcamtuf.coredump.cx/p0f.shtml`. Web-page retrieved 5th April 2011.

[22] William Stallings. *Cryptography and Network Security: Principles and Practices*. Prentice Hall, 4th edition edition, 2005.

[23] Eirik Holgernes and Øystein Aas Pedersen. How to Uncover Session Hijacking in Open 802.11 Networks. Technical report, Norwegian University of Science and Technology, 2009.

[24] Kai Tao, Jing Li, and Srinivas Sampalli. Detection of Spoofed MAC Addresses in 802.11 Wireless Networks. In *E-business and Telecommunications*, pages 201–213. Springer-Verlag, 2009.

[25] A. Bahillo, J. Prieto, S. Mazuelas, R. M. Lorenzo, J. Blas, and P. Fernández. IEEE 802.11 Distance Estimation Based on RTS/CTS Two-Frame Exchange Mechanism. In *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, 2009.

[26] Microsoft. About the Wireless Hosted Network. `http://msdn.microsoft.com/en-us/library/dd815243(VS.85).aspx`. Web-page retrieved 26th April 2011.

[27] DigiCert Incorporation. Phishing: A Primer on What Phishing is and How it Works. `http://www.antiphishing.org/sponsors_technical_papers/DigiCert_Phishing_White_Paper.pdf`. PDF retrieved 9th June 2011.

[28] Wireshark. About Wireshark. `http://www.wireshark.org/about.html`. Web-page retrieved 6th Mai 2011.

[29] The MadWifi Project. About MadWifi. `http://madwifi-project.org/wiki/About/MadWifi`. Web-page retrieved 14th December 2010.

[30] BackTrack. About BackTrack. `http://www.backtrack-linux.org/about/`. Web-page retrieved 23rd May 2011.

[31] Richard W. Hamming. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[32] fdXtended. HSMX - Internet Access Platform, Datasheet. `http://www.fdxtended.com/datasheets/HSMX-datasheet.pdf`. PDF retrieved 9th June 2011.

Appendix A

The Attacks

This appendix includes details of how the attacks can be performed in the Ubuntu environment. The attacks are also possible to perform in Windows, but the details are not included here. The first part of this appendix describes how the necessary information is obtained, then the next section describe the wait-for-availability attack. Some of the commands require sudo privileges, but the sudo prefix is excluded for simplicity.

A.1 Obtaining the Necessary Information

First, the attacker needs some basic information about the network in order to be able to start and sniff the network. To find the network name (SSID) and the corresponding channel the following commands are performed.

```
wlanconfig ath0 create wlandev wifi0 wlanmode sta    # (1)
ifconfig ath0 up                                     # (2)
modprobe wlan_scan_sta                               # (3)
wlanconfig ath0 list scan                            # (4)

# (1) Create a regular (sta) interface
# (2) Up the interface, that is make it available
# (3) Insert the scanning module to enable scanning for APs
# (4) Perform the actual scan and get the result as a list
```

The above commands should produce a list of wireless networks within range. With this list the attacker is able to find the desired network and its channel number. For *WirelessTrondheim* this was 6. Next, the attacker needs to configure the wireless NIC to be in monitor mode so that he can sniff the desired network, this is done with the following commands. These commands are the first four of the five commands shown in Section 4.2.3, nevertheless they are included here for completeness.

```
wlanconfig ath0 destroy                                # (1)
wlanconfig ath create wlandev wifi0 wlanmode mon  # (2)
iwconfig ath0 channel 6                                # (3)
ifconfig ath0 up                                       # (4)

# (1) Destroy any previous interface
# (2) Create a new interface in monitor mode
# (3) Monitor a specific channel, here channel 6
# (4) Make the interface accessible to the IP layer
```

When the NIC is correctly set in monitor mode the attacker can start Wireshark with the command `wireshark` and the monitor interface called `ath0` should be available for capturing. Assuming that there is at least one active client on the network the attacker should now see some traffic. Wireshark helps to identify the STAs by changing the prefix of their MAC address to the manufacturer that it belongs to. For instance the MAC address to an iPhone 1. generation (S-5) was shown as `Apple_03:d2:db`, the address to the LAP made by Cisco was `Cisco_0e:01:90` and the address of the Nomadix gateway as shown as `Nomadix_01:b5:eb`. The full MAC address is also shown in parentheses behind the manufacturer prefixed address. The attacker would then record the addresses of regular STAs (not LAPs or gateways) and wait for one of them to leave so that he can perform the wait-for-availability attack.

Before the attacker can perform the attack he must change the configuration of his NIC form monitor to sta. This is done by executing the same commands as above, except that second command is changed to `wlanconfig ath create wlandev wifi0 wlanmode sta`.

A.2 Performing The Wait-for-Availability Attack

Assuming the attacker has obtained the necessary information as described in Appendix A.1 he is now ready to perform the wait-for-availability attack. This is done by executing the following list of commands *after* he believes that the legitimate client has left. Determining whether the legitimate client has left can be done by either observing a special frame (Deauthenticate or Deassociate) from it or simply not seeing any traffic originating from target STA for a while.

```
ifconfig wlan0 down                                    # (1)
ifconfig wlan0 hw ether 00:1c:b3:03:d2:db      # (2)
iwconfig wlan0 essid "WirelessTrondheim"       # (3)
ifconfig wlan0 up 192.168.1.2                  # (4)
```

```
# (1) Put the interface down to be able to change the MAC address
# (2) Set the MAC address equal to the legitimate clients address
# (3) Set which wireless network to connect to
# (4) Up the interface and give yourself an IP to avoid IP conflicts
#     This connects the attacker to the given network
```

The attacker is now connected to the network, and he have the MAC address of a legitimate connected client, in this case an iPhone 1. generation (S-5). This should ensure that he bypasses the captive portal and gains access to the Internet, masquerading as the legitimate client.

Appendix B

The Scenario

B.1 Scenario 1, General Usage

Scenario 1 is a scenario performed by one single STA at a time without any other STAs connected to the LAP.

1. Connect to WirelessTrondheim (start timer)
2. Go to `www.vg.no` (dynamic site)
3. Open first story
4. Wait for one minute after the loading is complete
5. Go to `www.wikipedia.org`
6. Search for "Norway" (in English) and open that page (static page)
7. Refresh five times, wait until completely downloaded before refreshing again
8. Wait two minutes
9. Refresh again
10. Go to `www.vg.no`
11. Scroll all the way down while waiting for the timer to reach eight minutes
12. Refresh two times
13. Wait for one minute
14. Disconnect

Appendix C

The Source Code

C.1 FPP.pl

```
#!/usr/bin/perl
# Fingerprinting Program (FPP)
# by Christer Idland

use Net::Pcap;
use NetPacket::Ethernet;
use NetPacket::IP;
use NetPacket::TCP;
use Bit::Vector;
use IO::Handle;

use strict;
use warnings;
use diagnostics;

use constant {
    MGT => "00",
    CTRL => "01",
    DATA => "10",

    ASS_REQ => "0000",   # Type = MGT
    ASS_RES => "0001",   # Type = MGT
    BEACON => "1000",    # Type = MGT
    PROBE_REQ => "0100", # Type = MGT
    PSPOLL => "1010",    # Type = CTRL
    ACK_FRAME => "1101", # Type = CTRL
    CTS => "1100",   # Type = CTRL
    NULL => "0100", # Type = DATA
    QOS_NULL => "1100", # Type = DATA
};

```

```
my(
    $err,
    $pcap_err,
    $pcap_descr,
);

my $capture_file = "/home/christer/Dropbox/master//scenario captures/
    iPhone4G.cap";

my $ap_mac = "00235d0dfaa5";     # WrT Lap
my $gtway_mac = "0050e801b5eb"; # Nomadix

# Offset = 0 when only 802.11 frame is included and offset = 26 when
    radiotap headers is included
my $offset = 26;

my $in_offlinemode = 1; # Set to 1 to use the capture file, 0 for live
    capture

my %mac_signatures = ();
my %id_to_mac = ();
my $must_chk_next_pkt;

my %SrcSeq = ();          # Source MAC/Sequence Number combination
my %SrcAlm = ();          # Source MAC/Alarm Counter combination
my %qos_mac = ();         # QoS/MAC combination
my %cnt_mac = ();         # Counter and MAC combo
my %mac_state = ();       # Mac adress and 802.11 state

my $interface = "ath0";       # Defines interface for listening
my $snaplen = 1500;           # How much of each frame to be picked up
my $promisc = 1;              # Listen to all packets, Promiscous mode
my $readtimeout = 1000;       # Read timeout on packages
my $filecount = 0;

my $capture = ();

if($in_offlinemode){
    $capture = Net::Pcap::open_offline($capture_file, \$err);
    print"\n\nCapture has started (offline mode)\n\n";
}else{
    $capture = Net::Pcap::open_live($interface, $snaplen, $promisc,
        $readtimeout, \$pcap_err);
    print"\n\nLive capture has started \n\n";
}

my $w802 = Net::Pcap::datalink_name_to_val('DLT_IEEE802_11');
```

```
75 Net::Pcap::set_datalink($capture,$w802);
   Net::Pcap::loop($capture,-1,\&process_single_packet,'');
77
   unless (defined $capture){
79     die'Unable to create packet capture on device',"wlan1",' - ',
           $pcap_err;
   }
81
   Net::Pcap::close($capture);
83

85 sub process_single_packet{###-------====(( MAIN ROUTINE ))====-------###

87     my(@tim_aids);

89     my($pkt, $frame_ctrl, $dur_id, $dst_id, $src_id, $frametype,
           $subtype, $timestamp_m, $pwr_mgt) = extract_basic_info(\@_);

91     my $pkt_belongs_to_ntwrk = 1; # No need to filter

93     if($pkt_belongs_to_ntwrk){

95         ## Get additional info depending on type of frame ##

97         if($frametype == MGT && $subtype == ASS_REQ){
               my(@list_of_IEs) = find_info_elements($pkt);
99
               test_on_ass_req($pkt, $src_id, $dur_id, @list_of_IEs);
101        }

103        if($frametype == MGT && $subtype == ASS_RES){
               store_aid($pkt, $dst_id);
105        }

107        if($frametype == MGT && $subtype == BEACON){
               @tim_aids = get_info_from_beacon($pkt);
109        }

111        if($frametype == MGT && $subtype == 1100){
               print "DEAUTH\n";
113        }
           if($frametype == MGT && $subtype == 1010){
115            print "DEASS\n";
           }
117

119        ## Start tests ##
```

```
121         test1_use_pspoll($frametype, $subtype, $src_id, @tim_aids);

123         test2_use_keep_alive($frametype, $subtype, $src_id,
                $timestamp_m); #Lagrer ogsÃ¥ timedelta

125         test3_use_statechng_null($frametype, $subtype, $src_id,
                $pwr_mgt);

127         test3b_null_before_probe($frametype, $subtype, $src_id,
                $timestamp_m);

129         test4_use_fixed_interval($frametype, $subtype, $src_id,
                $pwr_mgt, $timestamp_m);
    }
131
}
133
sub extract_basic_info {
135
    my $_ = shift;
137     my ($data,$hdr,$pkt) = @$_;

139     # Gets unpacket in "wrong" endian, inside each byte.

141     my $data_rate = hex(unpack('H*',substr($pkt, 17, 1))) / 2;
                # Datarate from radiotap header

143     my $frame_ctrl = unpack('B*',substr($pkt, $offset + 0, 2));      #
            The 802.11 header starts from the 24th byte
        my $dur_id = unpack('B16',substr($pkt, $offset + 2, 2));         #
            The duration/ID field
145     my $dst_id = unpack('H*',substr($pkt, $offset + 4, 6));          #
            Destination address
        my $src_id = unpack('H*',substr($pkt, $offset + 10, 6));         #
            Source address
147     my $timestamp_s = ${$hdr}{'tv_sec'};                             #
            Timestamp in seconds for this packet
        my $timestamp_m = ${$hdr}{'tv_usec'};                            #
            Timestamp in microseconds for this packet
149
    my $timestamp = $timestamp_s + $timestamp_m / 1000000;
151
    my $frametype = substr($frame_ctrl, 4, 2);
153     my $subtype = substr($frame_ctrl, 0, 4);
    my $pwr_mgt = substr($frame_ctrl, 11, 1);
155
```

```
157     if(($frametype == CTRL && $subtype == ACK_FRAME) || ($frametype ==
            CTRL && $subtype == CTS) ){
            # ACK and CTS does not have source address, only RA address
159         $dst_id = $src_id;
            $src_id = 0;
161     }

163     return($pkt, $frame_ctrl, $dur_id, $dst_id, $src_id, $frametype,
            $subtype, $timestamp, $pwr_mgt);

165 }

167 sub test_on_ass_req {

169     # Frame body starts after 24 bytes of header

171     my ($pkt, $src_id, $dur_id, @list_of_IEs) = @_;

173     my $dur = bin2dec(substr($dur_id, 10, 8) . substr($dur_id, 0, 8));
        my $cap_list = unpack('B*',substr($pkt, $offset + 24, 2));
175     my $listen_interval = hex(unpack('H*',substr($pkt, $offset + 27, 1)
            ) . unpack('H*',substr($pkt, $offset + 26, 1)));

177     my $ass_req_rec = $mac_signatures{$src_id}{ass_req_rec};

179     my $length = scalar( @list_of_IEs);

181     if($ass_req_rec){

183         print "Has sendt an association request before, checking for
                consistency\n";

185         if($mac_signatures{$src_id}{ass_dur} ne $dur){ print "Duration
                is different\n";}
            if($mac_signatures{$src_id}{listen_interval} ne
                $listen_interval){ print "Listen Interval is different\n";}
187
            for(my $cnt = 0; $cnt < $length; $cnt++) {
189             my $old_value = $mac_signatures{$src_id}{$list_of_IEs[$cnt
                    ]->[0]};
                my $new_value = $list_of_IEs[$cnt]->[1];
191
                if($old_value ne $new_value){ print "IE with ID:
                    $list_of_IEs[$cnt]->[0] is different\n";}
193         }
```

```
    }else{ #No Ass Req has been recorded before for this MAC, so we
        need to save all the information

        $mac_signatures{$src_id}{ass_dur} = $dur;
        $mac_signatures{$src_id}{listen_interval} = $listen_interval;
        print "id: $src_id listen: $listen_interval\n";
        for(my $cnt = 0; $cnt < $length; $cnt++) {
            $mac_signatures{$src_id}{$list_of_IEs[$cnt]->[0]} =
                $list_of_IEs[$cnt]->[1];
        }
        # Set the ass_req_rec to 1 to ensure we check for
            inconsistencies next time we get a ass req
        $mac_signatures{$src_id}{ass_req_rec} = 1;
    }

}

sub find_info_elements {

    # ID, Startbyte (within ass req.), Length

    my $pkt = shift;

    my $moreIEs = 1;
    my $nextIE_pos = 28;
    my @list_of_IEs = ();
    my $total_length = length($pkt) - 4;     # Minus the Frame Check
        Sequence if radiotap is on

    while($moreIEs){

        my $id = bin2dec(unpack('B*',substr($pkt, $offset + $nextIE_pos
            , 1)));
        my $len = bin2dec(unpack('B*',substr($pkt, $offset +
            $nextIE_pos + 1, 1)));
        my $IE = unpack('B*',substr($pkt, $offset + $nextIE_pos + 2,
            $len));

        $nextIE_pos = $nextIE_pos + 2 + $len;
        $moreIEs = ($nextIE_pos + $offset != $total_length);

        splice(@list_of_IEs, 0, 0, [$id, $IE]);

        print "id: $id, len: $len, ie: $IE\n";
    }

    return @list_of_IEs;
```

```
235 }

237 sub store_aid {

239     my ($pkt, $dst_id) = @_;

241     # Ass res = header (24), capability info (2), status code (2), AID
            (2)

243     my $aid_field = unpack('B*',substr($pkt, 28 + $offset, 2));
        my $aid = bin2dec(substr($aid_field, 10, 6) . substr($aid_field, 0,
             8));
245
        $id_to_mac{$aid} = $dst_id;
247
        #Initialize fingerprint hashmap
249     $mac_signatures{$dst_id}{bcn_cnt} = 0;
        $mac_signatures{$dst_id}{use_pspoll} = 0;
251     $mac_signatures{$dst_id}{use_keep_alive} = 0;
        $mac_signatures{$dst_id}{null_cnt} = 0;
253     $mac_signatures{$dst_id}{statechng_cnt} = 0;
        $mac_signatures{$dst_id}{using_statechng} = 0;
255     $mac_signatures{$dst_id}{last_type} = 0;
        $mac_signatures{$dst_id}{last_subtype} = 0;
257     $mac_signatures{$dst_id}{null_probe_true_count} = 0;
        $mac_signatures{$dst_id}{null_probe_false_count} = 0;
259     #listen interval is initiated eariler
    }
261
    sub get_info_from_beacon {
263
        my $pkt = shift;
265
        # Frame body starts after 24 bytes of header
267     # TIM starts at 68 bytes (at least on beacon from WrT)
        # TIM is 6 bytes in length (but has TIM Length = 4) at lest when
            there a few (<8) users on
269
        my $bitmap = unpack('B*',substr($pkt, $offset + 73, 1));
271     my @bits = split(//, $bitmap);
        my $counter = 7;                    #bitmap of max id = 7 (0,1,2,3..)
            within a byte (8 bits)
273     my @tim_aids;

275     foreach my $bit (@bits){
            if($bit){ push(@tim_aids, $counter); }
```

```
277             $counter--;
        }
279     #print"bitmap: $bitmap, tim_aids: @tim_aids\n";

281     return @tim_aids;
}
283
sub bin2dec {   return unpack("N", pack("B32", substr("0" x 32 . shift,
    -32))); }
285
sub test1_use_pspoll {
287     my($frametype, $subtype, $src_id, @tim_aids) = @_;

289     # When frame == bacon frame check which AIDs that have data
            buffered (TIM)
        # Increment number of beacons with AID
291
        if($frametype == MGT && $subtype == BEACON){
293
            foreach my $aid (@tim_aids){
295             #print "Beacon frame tim_aids: @tim_aids\n";
                $src_id = $id_to_mac{$aid};
297             if($src_id){

299                 $mac_signatures{$src_id}{bcn_cnt}++;

301                 my $is_using_pspoll = $mac_signatures{$src_id}{
                        use_pspoll};
                    my $listen_interval = $mac_signatures{$src_id}{
                        listen_interval};
303                 my $bcn_cnt = $mac_signatures{$src_id}{bcn_cnt};

305                 if($is_using_pspoll == 1 && $bcn_cnt > $listen_interval
                        ){
                        print "MAC: $src_id went from using PS-Poll to NOT
                            using it (count: $bcn_cnt)! lstn interval:
                            $listen_interval\n";
307                     $mac_signatures{$src_id}{use_pspoll} = -1;
                    }
309             }
            }
311     }

313     if($frametype == CTRL && $subtype == PSPOLL){
            my $cnt = $mac_signatures{$src_id}{bcn_cnt};
315         #print "PS-Poll in use, (src_id: $src_id) count was: $cnt\n";
            if($mac_signatures{$src_id}{use_pspoll} == -1){
```

```
317             print "MAC: $src_id went from not using to using PS-poll\n"
                    ;
            }
319         $mac_signatures{$src_id}{use_pspoll} = 1;
            $mac_signatures{$src_id}{bcn_cnt} = 0;  # Reset beacon count
                after PS-Poll
321         #print "PS-Poll in use\n";
        }
323
}
325
sub test2_use_keep_alive{           # Check if STA sends keep alive
    Null data after a silent period of ca 10 sec (9>x<11>)
327
    my($frametype, $subtype, $src_id, $timestamp_s) = @_;
329
    if(!$mac_signatures{$src_id}{last_pkt_ts}){$mac_signatures{$src_id
        }{last_pkt_ts} = $timestamp_s;} #UnngÃ¥ undefined fÃ¸rste gang
331
    my $last_pkt_ts = $mac_signatures{$src_id}{last_pkt_ts};
333 my $time_delta = ($timestamp_s - $last_pkt_ts);
    $mac_signatures{$src_id}{last_pkt_ts} = $timestamp_s;   #update
        last pkt timestap
335 $mac_signatures{$src_id}{time_delta} = $time_delta;

337 my $use_keep_alive = $mac_signatures{$src_id}{use_keep_alive};

339 if((9 <= $time_delta && $time_delta <= 11) && isNullData($frametype
        , $subtype)){

341     if($use_keep_alive == -1){print "MAC: $src_id went from not
            using keep alive to using it!, delta: $time_delta\n";}
        $mac_signatures{$src_id}{use_keep_alive} = 1;
343
        print "Time delta: $time_delta - using keep alive mac: $src_id\
            n";
345
    }elsif($time_delta > 11 && defined($use_keep_alive)){ # && defined
        ensures that the AP is ignored as it has not been initialized (
        ass req)
347     if($use_keep_alive == 1){print "MAC: $src_id went from using
            keep alive to NOT using it!, delta: $time_delta\n";}
        $mac_signatures{$src_id}{use_keep_alive} = -1;  # Does not use
            keep alive
349
        print "Time delta: $time_delta - not using keep alive mac:
            $src_id\n";
```

```
351     }
}
353
sub test3b_null_before_probe{
355
    my($frametype, $subtype, $src_id, $timestamp_s) = @_;
357
    my $last_type = $mac_signatures{$src_id}{last_type};
359     my $last_subtype = $mac_signatures{$src_id}{last_subtype};

361     if($frametype == MGT && $subtype == PROBE_REQ){
        if(isNullData($last_type, $last_subtype)){
363             $mac_signatures{$src_id}{null_probe_true_count}++;
            print "Null before Probe ($mac_signatures{$src_id}{
                null_probe_true_count})\n";
365
        }elsif(!($last_type == MGT && $last_subtype == PROBE_REQ)){
367             print "NOT using null before Probe\n";
            $mac_signatures{$src_id}{null_probe_false_count}++;
369         }
    }
371
    if($mac_signatures{$src_id}{null_probe_true_count} == 30){
373         print "Using null before probe (30)\n";
        $mac_signatures{$src_id}{null_probe_true_count} = 0;
375     }

377     if($mac_signatures{$src_id}{null_probe_false_count} == 30){
        print "NOT Using null before probe (30)\n";
379         $mac_signatures{$src_id}{null_probe_false_count} = 0;
    }
381
    $mac_signatures{$src_id}{last_type} = $frametype;
383     $mac_signatures{$src_id}{last_subtype} = $subtype;

385 }

387 sub test3_use_statechng_null{

389     my($frametype, $subtype, $src_id, $pwr_mgt) = @_;
    my $time_delta = $mac_signatures{$src_id}{time_delta};
391
    pwr_state_based_on_bns($frametype, $subtype, $src_id);
393
    my $use_statechng_null = -1;
395
    my $set_size = 20;
```

```
397     my $hit_fraction = 0.85;

399     if($mac_signatures{$src_id}{must_chk_next_pkt}){
            # Know that last packet was Null Data with P = 0, changing
                state from P = 1
401
            if(!isNullData($frametype, $subtype) && $frametype == DATA){
403             $use_statechng_null = 1;
                $mac_signatures{$src_id}{using_statechng}++;
405             $mac_signatures{$src_id}{statechng_cnt}++;

407         }else{
            # Packet after Null that changed state is another Null or MGT/
                CTRL
409         # Ignore as we are only interested to know if the STA sends
                Null when it have data buffered
            }
411
            $mac_signatures{$src_id}{must_chk_next_pkt} = 0;
413
        }else{
415         if($mac_signatures{$src_id}{pwr_state}){     # in sleep mode
                before current packet
                if(isNullData($frametype, $subtype) && $pwr_mgt == 0){
417                 $mac_signatures{$src_id}{must_chk_next_pkt} = 1;
                }elsif($frametype == MGT || $frametype == CTRL){
419                 # Ignorere MGT og CTRL
                }elsif($frametype == DATA && $pwr_mgt == 0){ # Use data
                    packets, not Null to change state
421                 $use_statechng_null = 0;
                    $mac_signatures{$src_id}{statechng_cnt}++;
423             }
            }
425     }

427     if($use_statechng_null != -1){ #if it is -1 we need to wait for
            next packet before we can conclude

429         my $current_fpvalue;
            if(!defined($mac_signatures{$src_id}{use_statechng_null})){
                $current_fpvalue = $use_statechng_null;}
431         else{$current_fpvalue = $mac_signatures{$src_id}{
                use_statechng_null};}

433         $mac_signatures{$src_id}{use_statechng_null} =
                $use_statechng_null;
```

```
435         if($current_fpvalue != $use_statechng_null){
                $mac_signatures{$src_id}{tmp_teller3}++;
437             #print "MAC $src_id went from $current_fpvalue to
                    $use_statechng_null in using state chng null data\n";
                #print "changes: $mac_signatures{$src_id}{tmp_teller3}\n";
439         }
        }
441
        $mac_signatures{$src_id}{pwr_state} = $pwr_mgt;
443
        if(!defined($mac_signatures{$src_id}{statechng_cnt})){
            $mac_signatures{$src_id}{statechng_cnt} = 0;}
445
        if($mac_signatures{$src_id}{statechng_cnt} == $set_size){
447
            my $is_using = ($mac_signatures{$src_id}{using_statechng} >=
                $set_size * $hit_fraction);
449         if($is_using){print "Use state changing null, $mac_signatures{
                $src_id}{using_statechng} of  $set_size changes\n";}
            else{print "Does NOT use state changing null, $mac_signatures{
                $src_id}{using_statechng} of $set_size changes\n";}
451
            $mac_signatures{$src_id}{statechng_cnt} = 0;
453         $mac_signatures{$src_id}{notusing_statechng} = 0;
            $mac_signatures{$src_id}{using_statechng} = 0;
455     }

457 }

459 sub test4_use_fixed_interval {

461     my($frametype, $subtype, $src_id, $pwr_mgt, $timestamp_s) = @_;

463     my $buffer = 0.2;
        my $set_size = 20;
465     my $hit_fraction = 0.85;

467     if(!defined($mac_signatures{$src_id}{last_null_pwr})){
            $mac_signatures{$src_id}{last_null_pwr} = $pwr_mgt;}

469     if(isNullData($frametype, $subtype)){

471         if(!defined($mac_signatures{$src_id}{last_null_ts})){
                $mac_signatures{$src_id}{last_null_ts} = $timestamp_s;}

473         my $timestamp = $mac_signatures{$src_id}{last_null_ts};
            my $time_delta = $timestamp_s - $timestamp;
```

```
475         my $null_cnt = $mac_signatures{$src_id}{null_cnt};

477         if($pwr_mgt == 0 && $mac_signatures{$src_id}{last_null_pwr} ==
                1){
                # Only interested in a P=0 when P was 1 last time
479
                if($null_cnt == 10){
481                 my $average = $mac_signatures{$src_id}{average};
                    my $min = $average * (1 - $buffer);
483                 my $max = $average * (1 + $buffer);

485                 if($mac_signatures{$src_id}{pair_count} != $set_size){

487                     if(($min < $time_delta) && ($time_delta < $max)){
                            $mac_signatures{$src_id}{pair_ok}++;
489                     }
                    }else{
491                     my $pair_ok = $mac_signatures{$src_id}{pair_ok};
                        my $ok_fraction = $pair_ok / $set_size;
493                     if($ok_fraction >= $hit_fraction){print "Use fixed
                            interval, $pair_ok of $set_size par OK\n";}
                        else{print "Does NOT use fixed interval, $pair_ok
                            of $set_size par OK\n";}
495                     $mac_signatures{$src_id}{pair_count} = 0;
                        $mac_signatures{$src_id}{pair_ok} = 0;
497                 }

499                 $mac_signatures{$src_id}{pair_count}++;

501             }else{
                    $mac_signatures{$src_id}{delta_sum} += $time_delta;
503                 $mac_signatures{$src_id}{null_cnt}++;
                    #print "Delta til average: $time_delta\n";
505
                    if($mac_signatures{$src_id}{null_cnt} == 10){
507                     $mac_signatures{$src_id}{average} = $mac_signatures
                            {$src_id}{delta_sum} / 10;
                        $mac_signatures{$src_id}{pair_count} = 0;
509                     $mac_signatures{$src_id}{pair_ok} = 0;
                    }
511             }
            }
513
            $mac_signatures{$src_id}{last_null_ts} = $timestamp_s;
515         $mac_signatures{$src_id}{last_null_pwr} = $pwr_mgt;
        }
517
```

```
}

sub pwr_state_based_on_bns {

    my($frametype, $subtype, $src_id) = @_;

    if($frametype == MGT && $subtype == BEACON){
        for my $id ( keys %mac_signatures ) {

            $mac_signatures{$id}{bcns_since_data}++;

            if(defined($mac_signatures{$id}{listen_interval})){
                if($mac_signatures{$id}{bcns_since_data} >
                    $mac_signatures{$id}{listen_interval}){
                    $mac_signatures{$id}{pwr_state} = 0;
                    #print "Power state set to 0 due to beacons
                        exceeding listen interval id: $id\n";
                }
            }
        }
    }
    $mac_signatures{$src_id}{bcns_since_data} = 0;
}

sub isNullData{

    my($frametype, $subtype) = @_;

    if($frametype == DATA && ($subtype == NULL || $subtype == QOS_NULL)
        ){
        return 1;
    }else{
        return 0;
    }

}
```

FPP.pl