School of Electronics, Electrical Engineering and Computer Science

Final Year Project Report

Investigation of Trust and Reputation Metrics for 802.11 Hot Points

Matthew Hazley
13673041

11th May 2009

Supervisor: Professor Alan Marshall

Declaration

I declare that this document and its associated investigation are my own work and have been generated solely as a result of my own research.

Signed: _____ Date: _____

Acknowledgements

Without certain individuals this project would not have been possible.

I would like to thank Prof Alan Marshall for his open mindedness regarding the project specification initially and also for his friendliness, support, expert guidance and keen interest throughout the year.

I also wish to express my gratitude to Alistair McKinley. Without his adept programming skills and “guru” knowledge of Linux this project would not have been possible. I also thank him for his idea to use libpcap.

I would like to thank Jonny Milliken, whom I developed the attacks with, also.

Finally, I would also like to thank my close friends and family who supported me across the year.

Abstract

Ubiquitous 802.11 networks are becoming more and more common each and every day due to the popularity of the open connectivity and easy access to services they provide. While Wireless LANs in general offer a number of specific insecurities, the open nature of ubiquitous WLANs leave them especially vulnerable to malicious users. As such, there is increased interest in the ability to determine the trustworthiness of a network.

This project attempts to investigate trust and reputation within the ubiquitous WLAN by first understanding how malicious users can present themselves on the network. A packet capture tool is then developed in C to promiscuously capture power values from frames transmitted from APs. Average power and power variance metrics are then calculated from these captured values and the metrics are monitored over time.

The tool is tested under malicious attack scenarios and the metrics are shown to display highly observable characteristics that do seem to indicate when an AP has become untrustworthy. This is especially interesting as power metrics are not used in any in-the-field applications today.

This being said, classification of these results is deemed to be required so as to make this decision of trust.

Table of Contents

1 Introduction

Wi-Fi networks have thrived in modern society and continue to grow in a large and increasingly “open” manner. They are an inherently insecure form of communication and they demonstrate certain characteristics that make them significantly different from traditional wired networks. This is especially true for Wi-Fi networks that are open to the public or are easy to gain access to.

These kinds of networks are increasingly popular in high density public situations such as train stations, coffee shops, airports etc and with the ever increasing popularity of mobile computing devices with 802.11 connectivity, such as laptops and PDA’s, the use of these networks is becoming progressively increased.

This being said, one point that requires understanding is that these so called open wireless networks are inherently different to, say, a home or corporate wireless network. Often, they have no provisions for wireless security, and if these are in place they are usually openly known or vulnerable to attacks. In essence, the networks sole purpose is to allow open connectivity to any party.

In these situations, the network becomes increasingly vulnerable to a wide range of attacks from malicious and rouge users. These individuals may wish to attack the network and its legitimate users, or draw unfair amounts of resources away from them (i.e. selfish behaviour). There are a number of ways in which this can take place including such attacks as Denial of Service, Evil Twin/Rogue Access Point and man in the middle.

At the same time, it is the very open nature of these networks that provides their benefit; they permit users to ubiquitously access the Internet and services. As connectivity becomes more pertinent in today’s society, this offers an advantageous and attractive prospect for modern users wish to stay connected at all times.

Therefore, if these networks are to grow and mature, the design of techniques to help identify reliable wireless networks and wireless devices presents a major challenge and one which is important in order to maintain the open nature of these networks.

Recently, there has been growing interest in determining and utilizing trust and reputation metrics that can be used to identify and extrapolate malicious users and those that might abuse and misuse a network and its legitimate clients. It is important that genuine users are able to access these networks in a trusted manner.

2 Project Specification

Below is the original project specification.

Wireless networks are an inherently insecure form of communication; they have fundamental characteristics that make them significantly different from traditional wired networks. This is particularly true for wireless networks that are open to public access such as WLAN hot points, where rogue access points ("Evil Twins") can fool users into thinking they are connected to legitimate networks. Ad-hoc networks have similar problems. At the same time, it is the very open nature of these networks that provides their benefit; they permit users to ubiquitously access Internet services. Therefore the design of techniques to identify reliable wireless networks and wireless devices presents a major challenge. Recently there has been a lot of interest in peer-to-peer (P2P) schemes that create trust and reputation metrics that can be used to identify reliable networks and devices from those peers which might abuse and misuse the networks and legitimate clients.

The objectives of the project are:

1. *Identify different methods how malicious peers and rogue access points can appear as legitimate devices in a network.*
2. *Investigate the different methods for implementing trust and reputation among P2P systems.*
3. *Develop a simple algorithm that can be used to generate reliable reputation metrics about WLAN access points and their users.*
4. *Implement your trust and reputation scheme in either a network simulator, or an actual network, and show how it can correctly discriminate between reliable and unreliable nodes/ access points.*

MEng Extensions

5. *Extend your trust and reputation scheme to cover a larger network which has several access points, and show how reputation metrics can evolve and change over time.*
6. *Examine the performance of your trust and reputation scheme in terms of accuracy, efficiency, cost etc.*
7. *Undertake an assessment of other commercial counter measures against these types of attack and contrast against your system.*

1.1.1 Learning Outcomes

1. *Understand how attacks against WLANs work.*
2. *Be able to design the algorithms and software required to implement this type of management scheme.*
3. *Be able to set up the system, test it, examine and analyse its performance.*

2.1 Main Objectives

The specification details the problem essentially quite similarly as was described in the introduction; that open Wi-Fi networks provide substantial benefit in offering ubiquitous connection to the user but that the very nature of them leaves them substantially vulnerable to attack from malicious users.

As can be seen, the specification suggests that some investigation should be carried out into determining metrics that will enable the determination of trust and reputation on these open Wi-Fi networks.

The main objectives for the project can be detailed as follows:

- Investigate the different methods for implementing trust and reputation among P2P systems.
- Identify different ways in which malicious peers and rogue access points can appear as legitimate devices in a network.
- Develop a simple algorithm that can be used to generate reliable reputation metrics about WLAN access points and their users.
- Implement your trust and reputation scheme in either a network simulator, or an actual network, and show how it can correctly discriminate between reliable and unreliable nodes/ access points.
- Evaluate the viability of this solution.

A number of additional objectives that can be considered as extensions to the main objectives are also detailed from the specification.

- Implement trust and reputation scheme on a realistic open network deployment.
- Investigate how the trust and reputation metrics can evolve over time.
- Examine the performance of the solution in terms of accuracy, efficiency and cost.
- Assess other countermeasures against these attacks and discuss the contrast.

Following some initial investigation and planning, a Gantt chart will be put in place to give an overview of the projects main tasks and timeframe.

3 Background Research into Trust and Reputation

One of the initial objectives of the project is to carry out some background research into a number of different areas in order to gain some form of perspective for the project and to enable more focused aims and objectives to be put in place.

A number of different research areas could potentially be examined within the scope of this project and below are brief overviews of some papers and journal articles that detail work within both the trust and reputation field and also within the distributed network security field. As mentioned in the specification objectives, Peer to Peer (P2P) networks concern themselves heavily with both trust and reputation and as such, may provide this project with a number of interesting insights.

3.1 Taxonomy of Trust: Categorising P2P Reputation Systems

Sergio Marti & Hector Garcia-Molina [1]

This article is a cut down summary of Hector Garcia-Molina's larger paper on the Taxonomy of Trust and as the title suggests, it concerns the categorisation of peer to peer reputation systems. The paper essentially manifests itself as an excellent break down of current P2P systems and the general methods that they employ alongside analysing the behaviour of users, both malicious and legitimate. It then attempts to describe some issues and compromises that arise when designing and delivering usable systems.

Some of the main points that this article make are very helpful and insightful when considering implementing some form of trust system and a number of these are detailed below.

Molina states that it is initially important to break down the functionality of such a system into three components:

- Information gathering
- Scoring/Ranking
- Response

Essentially what Molina is saying here is that the system must have some way to gather the information upon which it will make a trust/reputation decision. Based on this decision, some form of response can be carried out.

He also distinguishes clearly between the types of adversary that could appear within a system and divides these into malicious users and selfish users. Malicious users' sole aim is to cause harm to the system or the target, rendering it useless whereas selfish users are there to use all the resources and never contribute or share. Molina

makes the point that it is important for a system to tackle a certain kind of adversary as no one system can realistically tackle them all.

He extends this *threat model* by continuing to state that it is important to examine the ways in which these malicious users can attack the system and the tactics that they might employ as this will continue to form the basis for the types of behaviour that the system can work towards targeting.

Following this, one of his most important statements is to simply say that a reputation/trust decision can only be based on a strong pool of collected information that can be understood and acted upon.

Once this information is pooled and acted upon, it is important that calculation provides some form of output to determine a level of trust/reputation and this can take a number of forms, the most promising is that a trust threshold can be determined and measured against. Molina also states that it is considered advantageous if a number of different calculated values can be used to make the final trust/reputation decision.

Obviously, a P2P system is quite different from a ubiquitous 802.11 connection but the important thing to draw from this article is that the core ideas of a general trust/reputation system can play a large part in beginning to understand and develop such a system for any purpose.

3.2 CORE: A Collaborative Reputation Mechanism

Pietro Michiardi and Refik Molva [2]

This paper was studied as part of the investigation into the trust/reputation schemes employed by Ad Hoc networks, especially Mobile Ad Hoc Networks (MANETs). CORE is a mechanism based on reputation that works to encourage cooperation between nodes on Ad Hoc networks. It forms reputation based on nodes' experiences of other nodes and the information that they gathers.

Ad Hoc networks are unlike the proposed networks in this project in that they have no centrally dedicated node involved in routing and this leads them to become more sensitive to malicious nodes.

CORE is a relatively complex statistical analysis scheme that essentially stores reputation data in each node on the Ad Hoc network. Each node monitors its neighbour's activity and using what is known as a Watchdog Mechanism, it evaluates whether this is considering detrimental activity and updates that nodes reputation data and forms a trust metric based on this. This means that a node must cooperate or it will run the risk of becoming untrustworthy and the network will essentially lock it out.

As mentioned, while this is essentially quite different to an 802.11 infrastructure scenario, potential ideas can be gained from this concept. In any trust/reputation scheme it is important for information to be monitored, recorded and stored in order to form a decision of trust and this is a decision that can possibly be made individually by each node that is present on the network.

3.3 Addressing Wireless Threats in the Cisco Unified WLAN

Cisco White Paper [3]

This paper was examined in the hope that some interesting parallels could be drawn from the corporate methodology of dealing with wireless threats and then these could be mirrored in a ubiquitous connection. The opposite of this occurred though and it was found that corporate method of carrying this out is much too stringent for an open situation and is also really only applicable to an entity with a certain number of access points in one place.

Essentially, Cisco employ a technique that involves both constant monitoring of their wirelessly covered area by wireless sniffers and they also employ a technique known as Management Frame Protection (MFP).

For the ubiquitous situation, the employment of sniffers linked to a central management system to search for unauthorised activity is highly unfeasible and essentially removes the convenience and self manageability from ubiquitous connections. It's something that large companies can afford to implement and have the resources to manage centrally.

The MFP that is employed works towards eradicating many of the threats based around spoofing management frames. It involves applying a hash to the frame which can be used as an integrity check at the client. Again, this is an unfeasible line of defence in ubiquitous connections as it requires special equipment, central management and all clients to be equipped with specific software beforehand. These are all things that renounce the ubiquitous nature of the connection in question in this project.

While this paper doesn't shed any light on new techniques that could possibly be employed it does provide information on tactics that are outside the scope of the project in question and allows the perspective of the investigation to become more focused.

3.4 Summary of Research

The three papers detailed here offer a number of ideas for implementing a system of trust/reputation and a number of key points can be drawn from them.

A Trust/Reputation system must first gather information and this must be based on the expected malicious behaviour of adversaries on the network. It also must be gathered in a way so as to not disrupt normal network behaviour as this will be detrimental to the system. Once this information is gathered, something must be done with it in order to determine its trustworthiness and this could be as simple as weighting a calculated value against a trust threshold. It is also important to note here that individual nodes on the networks themselves could be potentially employed to carry out this information gathering.

Some form of resultant response can be carried out by a node or the network based on the information that has been gathered.

4 Background Theory

In order to begin to investigate the implementation of a trust metric in a ubiquitous 802.11 network, there is a certain element of background theory that must be investigated and understood initially in order to provide clarity and understanding. This is essentially a look at the 802.11 technical specifications with some additional study into 802.11 information headers and network packet capture.

4.1 IEEE 802.11 (Wi-Fi)

IEEE 802.11 is basically a specification for wireless local area networks (WLANs) and is developed by a working group of the Institute of Electrical and Electronics Engineers (IEEE). It is more widely known as Wi-Fi and is a well known and popularly used protocol for wireless networking of personal mobile devices.

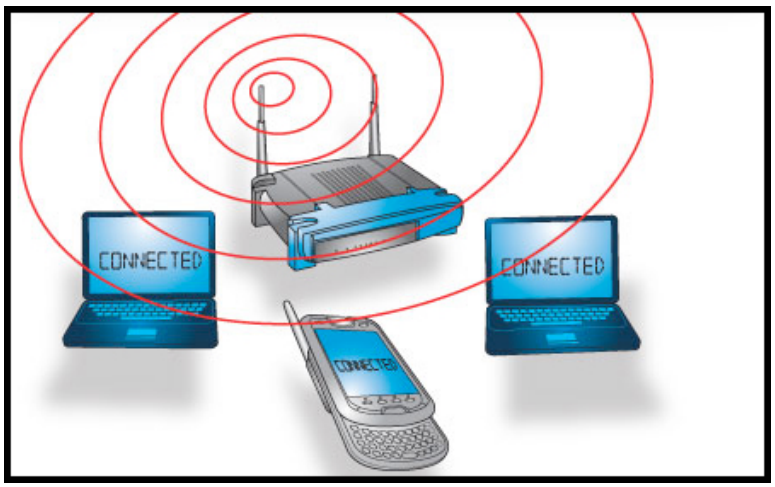


Figure 1 - A typical Wi-Fi deployment

Within this project, the ubiquitous network being investigated utilises the 802.11 network protocols so it is important to have an understanding of certain theoretical components of the protocol as this will provide assistance later when dealing with a networks operation.

4.1.1 CSMA/CA

As with any of the IEEE 802.x protocols, the 802.11 protocol specifies details for a MAC layer and a Physical layer. The MAC layer basically describes two different ways to access the medium and the most commonly used is the Distributed Coordination Function. This is basically Carrier Sense Multiple Access with Collision

Avoidance (CSMA/CA) which is almost exactly comparable to the well know Ethernet standard that operates CSMA with Collision Detection as opposed to avoidance.

The way in which Collision Avoidance works is in cooperation with a scheme known as Positive Acknowledge and essentially, the following takes place:

- A station willing to transmit senses the medium and if busy, it defers.
- If the medium is free for a certain length of time (known as the Distributed Inter-Frame Space [DIFS]) then the station is allowed to transmit.
- It first transmits a RTS (Request to Send) packet and if the medium is free, the destination station will respond with a CTS (Clear to Send) packet.
- These packets contain duration information which allows the medium to be reserved for a transmission time (by setting something called the Network Allocation Vector [NAV])

This mechanism works well and provides what is known as Virtual Carrier Sense and reduces the probability and overhead of collisions. The following diagram displays this concept diagrammatically. [4]

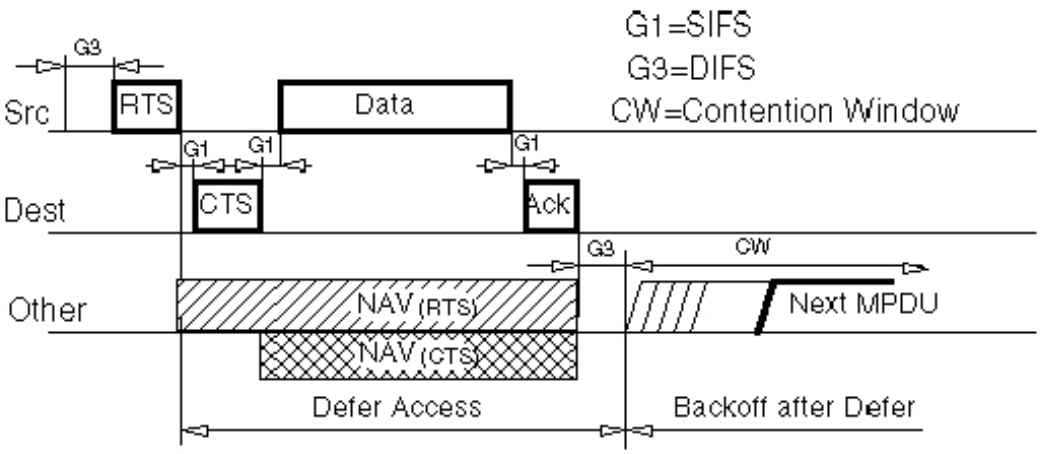


Figure 2 - Virtual Carrier Sense Mechanism[4]

4.1.2 Packet Structure

Before examining the 802.11 packet structure, it is important to make a comment on fragmentation. In a standard LAN deployment utilising Ethernet for example, a packet could be anything up to 1518 bytes long. When operating a wireless LAN deployment on the other hand, there are a number of reasons why it would be advantageous to utilise smaller packet sizes. For example, a radio link has a higher bit error rate and there is a higher chance of a packet getting corrupted if the packet size is larger and this also means that there is less overhead if a retransmission must take place.

The following figure displays a frame being divided up into several fragments, known as MPDUs.

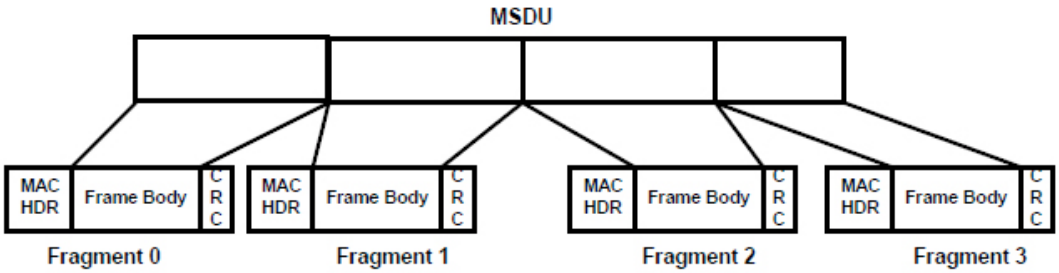


Figure 3 – Fragmentation[4]

All 802.11 frames are composed in the following manner:

Preamble	PLCP Header	MAC Data	CRC

Figure 4 - 802.11 Frame

The preamble is dependent on the physical layer and it contains synchronisation and timing details for the transmission while the PLCP Header contains more information for the physical layer to assist in decoding the frame.

The MAC data section of the frame is possibly the most important aspect of the frame as this is where all of the MAC information is carried and essentially where the purpose of the frame is depicted. The following diagram shows the MAC Frame Format:

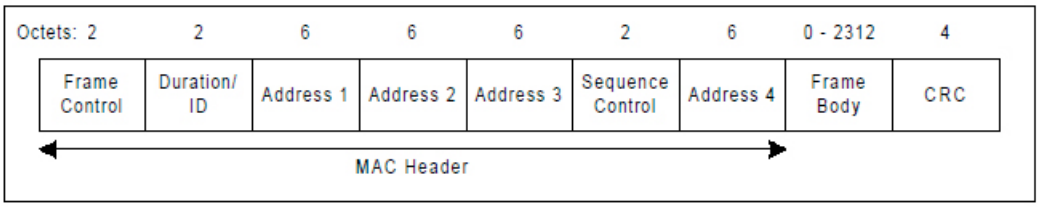


Figure 5 - The MAC Frame Structure[4]

As can be seen, the initial 30 bytes of this is the MAC Header and it contains information about the frame itself as well as its destination and location.

The following diagram displays the structure of the Frame Control section of the header as it has numerous pieces of information spread over its two bytes.

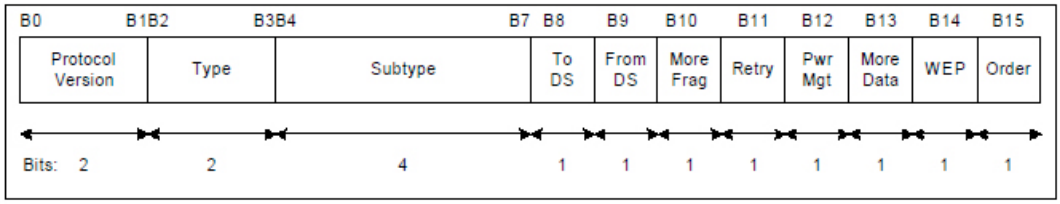


Figure 6 - The Frame Control Section[4]

The important aspects of the MAC Header are detailed below with a description of the information that each field holds.

- **Frame Control**
    - **Type and Subtype:** This determines the type of the frame (i.e. Management, Control or Data) and the subtype describes the kind of frame within that type.
    - **ToDS & FromDS:** The ToDS bit is set to 1 when the frame is addressed to the access point and is 0 in all other cases. The FromDS bit is set to 1 when the frame is coming from the distribution system.
    - **Retry:** This field indicates whether the frame is a retransmission of a fragment.
- **Duration/ID:** This is the duration used to calculate the NAV that was described in the initial Carrier Sense description.
- **Address1:** This is always the recipient address – if ToDS is set then this is the address of the Access Point and vice versa.
- **Address2:** This is always the Transmitter Address – if FromDS is set then this is the address of the Access Point.
- **Sequence Control:** This is used to represent the order of fragments that come from the same frame. [5]

4.1.3 Wi-Fi State Machine

It is important that the Wi-Fi state machine is detailed as it describes the way in which stations authenticate and associate themselves with an Access Point in order to fully connect and use the medium as a node on the network.

Essentially, a station that wants to join a WLAN must first discover the WLANs that are present. This is accomplished in two ways; passively, meaning that the station must listen for a Beacon which is a type of management frame sent by the Access Point, or actively, which means that the station transmits a Probe Request on all available channels, until it receives a Probe Response from the Access Point. The Beacon and Probe are simply two types of management frames.

Once a WLAN is decided upon for connection, the station enters what is known as the Wi-Fi state machine and this, as the name suggests, represents a number of states that are used when connecting and disconnecting from an Access Point. The diagram below depicts the state machine.

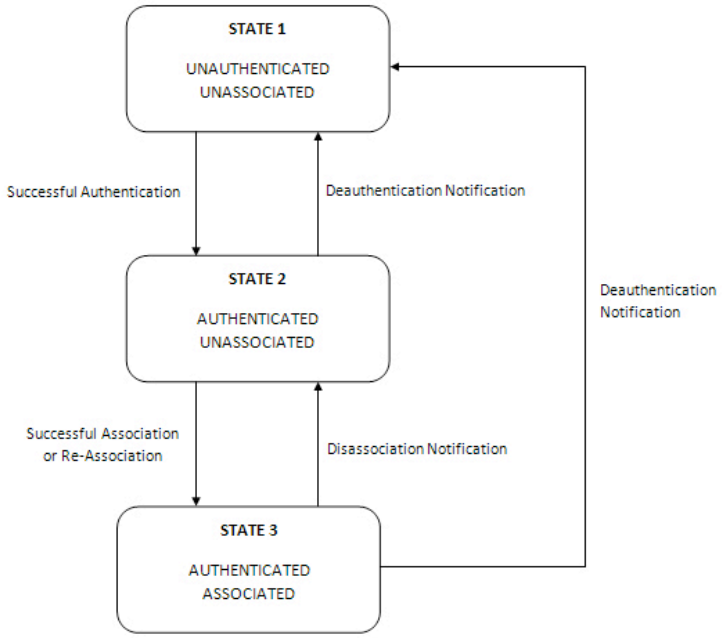


Figure 7 - The Wi-Fi State Machine

As can be clearly seen, the station must go through an authentication stage and then an association stage in order to become fully authenticated and associated with an Access Point. When this occurs, the station is connected to the Access Point. [5]

The state machine is fundamental to understanding some of the attacks that take place against WLANs and this will be discussed further later in this report.

4.1.4 802.11 Channels

Wi-Fi operates within the 2.4GHz band which is deemed unlicensed spectrum by international agreement. The diagram below shows the 14 channels that are in operation and their frequencies of operation.

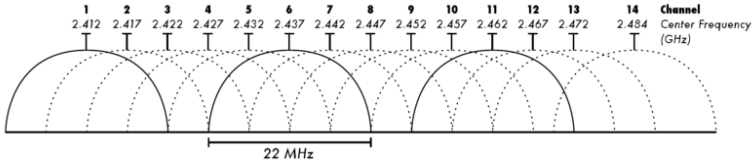


Figure 8 - 802.11 channels

Different countries have different specifications as to how their 802.11 deployments must operate and some use different numbers of channels. In the UK and USA, channels 1-11 are permitted for operation and it is recommended that channel 1, 6 or 11 is used. This is simply suggested as this minimises interference from adjacent channels as can be seen from the above diagram. [5]

4.2 Additional 802.11 Headers

When utilising a WLAN connection, the device in question will have some form of actual hardware to enable it to communicate using the 802.11 protocols. This device will ultimately be controlled by a software driver in the operating system, and with modern drivers, additional information is sometimes transmitted alongside the 802.11 frames in the form of a small header.

These can take a number of forms and essentially, they are a clever way for the driver to send extra information about the packet. For example, the 802.11 header does not carry any information regarding the signal strength of a transmission yet this may be information that is desirable. Given that information gathering is ultimately one of the key factors when developing a trust/reputation based system, this information may be extremely useful.

There are a number of different kinds of 802.11 headers used by a wide range of different drivers but the two most commonly used are Prism and Radiotap and a short breakdown and detail of each are included below.

4.2.1 Prism Header

Drivers that support Prism headers will attach a fixed length structure that is 144 bytes in length. This additional header contains received signal strength (RSSI), capture device, channel, and other signal/noise quality information that may be useful to the user. [6]

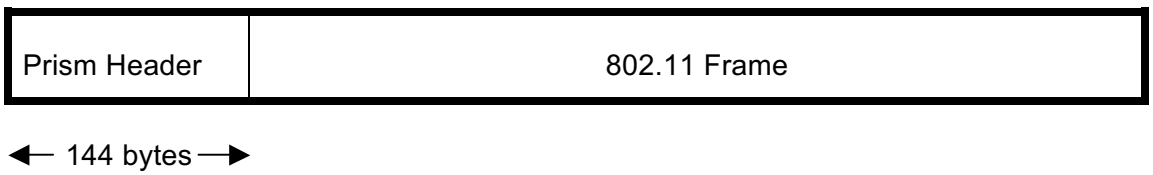


Figure 9 - Prism Header

4.2.2 Radiotap Header

The Radiotap header is a flexible format that is not a fixed length like the Prism header. The developers claim that this provides more flexibility and allows the driver developer to specify an arbitrary number of fields as they choose.

How this works is that the structure that is appended to the frame by the driver is made up of two sections; the Radiotap header and the Radiotap data. The Radiotap header is of fixed length and Embedded within it is the length of the entire Radiotap structure and also a 32 bit mask. This 32 bit mask specifies which data fields follow the header and it is used by a parser wishing to read the data. As the Radiotap structure is of variable length, the length and mask are important as the parser must know how long the section of data is and what fields are present. [7]

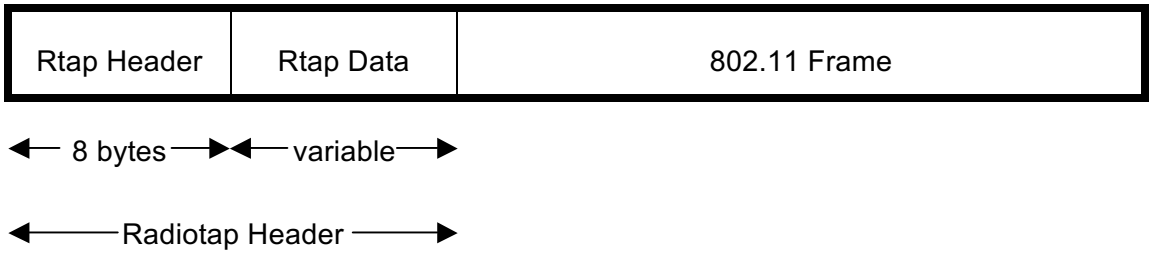


Figure 10 – Radiotap Header

Radiotap currently carries similar information to Prism and has the possibility to detail such things as Antenna information, transmission power, RSSI etc. Drivers that utilise the standard have a choice as to which information they wish the carry.

This, in a way, makes Radiotap quite advantageous for a number of reasons. First, certain drivers can contain more or less information and this gives the developer/user a choice based on the requirements that they have. Secondly and possibly most importantly, is that the standard is of variable size and can easily be expanded to contain more fields and carry more information in the future.

4.3 Packet Capture

Packet capture (sometimes known as sniffing) is simply the act of capturing data packets crossing a network. Once captured, tools can be used to review packet data, perform analysis, identify security threats, and ensure data communications and network usage complies with outlined policy.

Packet capture can be classified as complete capture or as filtered capture. Complete capture involves the capture of all packet data including headers and payloads. Complete capture is considered to be the unrestricted raw capture of all network packets. Filtered capture simply details the application of filters to the above idea in order to only capture packets that meet a certain criteria.

Once data is captured it can either be stored in a capture file for later analysis or it can be analysed in real time as the packets arrive. This is specific to the application looking to utilise the packet data.

There are a number of currently employed uses for packet capture, ranging from network security to network analysis and diagnostics. It can be used to indentify instances where unauthorised intrusions occur, troubleshoot network problems, lawfully intercept traffic, benchmark network performance and verify security fixes.

These are just a number of popular applications. Essentially, if a use can be found for the data that packet capture provides, then the concept of packet capture can be integrated to many different applications and situations.

5 Project Approach

As with a large proportion of projects, a general specification is detailed at the beginning stages in order to form a basis for the thought development and initial investigation process. It is almost inevitable that both thinking and approach will change as the project progresses.

Based on the investigation that has taken place, the main objectives of this project haven't changed, but the direction and core aims of the project have become more apparent and focused, allowing the objectives to be detailed in greater depth.

One of the main initial objectives will be to investigate the ways in which malicious peers and rogue access points can appear as legitimate devices in a network. This backs up the claims made by Hector Molina in his paper in which he states that it is important to develop a threat model by understanding the ways in which malicious peers operate on a network and the kind of activities they carry out.

In order to investigate this fully, a range of malicious attacks will be carried out on an 802.11 network in a laboratory environment so that the methods and behaviour that a malicious user would adopt can be investigated and understood.

The next step is to begin thinking about how to develop some way of forming a trust/reputation decision on a network and to implement this algorithmically.

Referring again to Hector Molina's taxonomy of a trust system, he states that the core initial phase of any system is to carry out information gathering and to rank this in some way. In this case, it will be important to develop some sort of packet capture program in order to obtain information from the network and somehow utilise this to form a decision on whether aspects of the network can be trusted.

This means that as the behaviour of an attacker is understood by developing the attacks as mentioned, the information obtained from the packet capture can be properly utilised to form a trust decision.

This packet capture program can then be developed to capture the specific information required and provide a means to making a decision on trust.

5.1 Refined Objectives

For clarity, an overview of the 1more focused objectives can be found below. As can be seen, these do not deviate completely from the initial set of general aims but merely form the basis of a more concentrated and focused investigation.

- Investigate and simulate, in a laboratory environment, a range of 802.11 attacks and malicious activities
    - Denial of Service (DoS) Attack
    - Crack WEP/WPA Security
    - Rogue Access Point/Evil Twin Simulation
    - “Man in the Middle” attack
- Develop a Packet Capture program using the C programming language.
- Investigate the feasibility and potential for utilising the captured data as a means of creating metrics to determine the trustworthiness of an Access Point.
    - Investigate the use of additional header data
- Carry out experiments with this captured information on a physical and realistic network distribution to determine the viability and usefulness of its potential use as a trust metric.

In appendix 3, the Gantt chart produced for the duration of this project can be found detailing the key tasks and providing an overview of a timescale for each.

6 Software Tools

In order for a lot of the tasks detailed in the objectives to be carried out a number of different software tools are required in order to run experiments and carry out simulations. The following is a short description of each software tool, provided in an attempt to clarify their use within the project.

6.1 Linux

Linux is a generic term referring to Unix-like computer operating systems based on the Linux kernel. Their development is considered one of the most well-known examples of free and open source software development and collaboration. Typically all the open source code available can be used, freely modified, and redistributed by anyone under the terms of a number of free software licenses.

There are a number of reasons for utilising Linux as opposed to a more popular operating system such as Windows in this project. Firstly, as mentioned above, the "open source" nature of the environment means that's it is easy to acquire the applications and tools that are required to carry out attacks and to experiment with things like wireless injection and sniffing. As mentioned, the project will include a certain amount of software development regarding the creation of a packet capture program and one of the most productive ways of learning about the coding of this is by examining other software that carries out a similar function. Normally, with Windows, this wouldn't be possible but the beauty of the open source ethos is that it openly encourages this as a means of learning and development.

Most of the tools that will be used for carrying out attacks are small, open source packages and sometimes these may even be little "home-brew" pieces of code from universities or individuals. These are easily compiled and utilised for Linux and in most cases, will already be offered as an installable package.

Linux distributions normally support many different kinds of programming languages straight out of the box, which make it an excellent operating system for attempting some low key development on. This is especially important in the case of this project as whereas C will mainly be used, numerous libraries will have to be utilised alongside this and certain other scripts may need to be compiled in different languages. Essentially, Linux gives the freedom required here.

Also, Linux provides support for unofficial, open source device drivers which can be especially useful when attempting to carry out wireless attacks. Drivers of this nature enable wireless hardware to be used in numerous ways, such as monitor mode, which can be useful for sniffing and injecting packets.

Essentially, Linux offers two things that are required of this project and these are namely simplicity and flexibility.

6.2 Atheros/MadWiFi

When Linux began to become popular, it became apparent that not all hardware vendors were especially adept in supporting it as an OS. Thus, a lot of the open source projects in existence were there to create drivers and support for different hardware. One such group is called the MadWiFi group.

MadWiFi is short for Multiband Atheros Driver for Wireless Fidelity. Essentially, it is a Linux device driver for WLAN hardware based on Atheros Technology. While allowing the WLAN card to appear as a normal network interface in the system, there is support for what is known as the Wireless Extensions API which is a generic API allowing a driver to expose configuration and statistical information about the WLAN to the user-space. This means that most aspects of the device can be viewed and configured.

MadWiFi is considered to be one of the most advanced WLAN drivers available for Linux today. A large number of functions are included, enabling the wireless NIC itself to run in a number of different modes, some of which are not only helpful but are essential for carrying out attacks and packet capture. Another key feature of the driver is that it is possible to run a number of different connection interfaces on one card. This way, a computer can connect to an AP while acting like an AP for a number of other machines for example, which, as will be discussed, is the basis of some very effective attacks. [8]

The driver carries this out by implementing a parent wireless device called *wifi0* and then it implements Virtual Access Points (VAPs) on top of this which it calls *ath0, ath1* etc. Each of these can operate in a specified mode.

This driver also supports the Radiotap header as an additional appendage to the frame. This means that the frame carries non-specific 802.11 information that can be potentially analysed.

6.3 Aircrack-ng

Aircrack-ng is a collection of open source software that consists of a detector, packet sniffer, WEP and WPA cracker and analysis tool for 802.11 wireless LANs. It works with any wireless card with a driver that supports monitoring mode and as such, it works well with the MadWiFi driver discussed above.

Essentially, the main aspects of the tool that will be utilised within this project are as follows.

6.3.1 Aircrack-ng

This is a WEP/WPA cracking tool that utilises a pool of specific pre-captured frames and uses these alongside a number of statistical and brute force methods to determine a security key.

6.3.2 Aireplay-ng

The main purpose of Aireplay-ng is to inject frames. Its main function is to generate traffic that will enable specific frame capture for the statistical attacks in the Aircrack-ng tool. This being said, it can also generate a number of attacks such as deauthentication, fake authentication and ARP request injection.

6.3.3 Airodump-ng

This aspect of the tool is used to collect raw 802.11 frames. It is essentially a packet capturer and is mostly used to collect frames that are used in the Aircrack-ng attack. It also provides the user with a textual output of all access points and users seen.

6.3.4 Airbase-ng

This is an extremely versatile tool that is still currently under development somewhat. Its main purpose is to create a fake AP that it will encourage clients to connect to in order to attack them. Essentially, it listens for probe requests from clients and replies with appropriate response, encouraging clients to authenticate with the Airbase AP. This can be used as the basis of a number of attacks such as man in the middle and the rogue access point.

6.4 libpcap

Libpcap is an open source programming library that essentially provides a high level interface to packet capture programs. It provides a number of detailed functions that allow the user to essentially capture any packet on a network, even those that are destined for other hosts.

The main objective of libpcap is to create a platform-independent way of implementing packet capture that eliminates the need for system dependency. The API is designed to be used in C/C++ and as libpcap will be used to implement a packet capture tool later, some elements of the API will be further explained as they are used.

6.5 Other Tools

Some additional tools that have and will be utilised throughout the project are as follows.

D-Sniff is a suite of tools that contains software to transmit fake ARP messages to the victim machine. Ettercap is a suite of plug-ins that can be used on sniffed packets as they are forwarded through the machine in a MITM attack. DHCP3 is an emulating DHCP server that needs to be used to enable connectivity to a virtual access point.

7 802.11 Attacks

As mentioned, before implementing any form of trust based system, it is vitally important to understand fully the threat model that exists within the system being monitored. In the case of open 802.11, the threat model can be developed by investigating and simulating a number of attacks that could be potentially carried out by malicious users. Understanding these attacks will then allow certain behaviour to be examined and this will provide the basis for investigating trust metrics.

The following attacks will be investigated:

- Denial of Service
- Cracking WEP/WPA
- Rogue Access Point/Evil Twin
- Man in the Middle

7.1 Denial of Service

The following diagram displays a DoS attack much like the one that is carried out below.

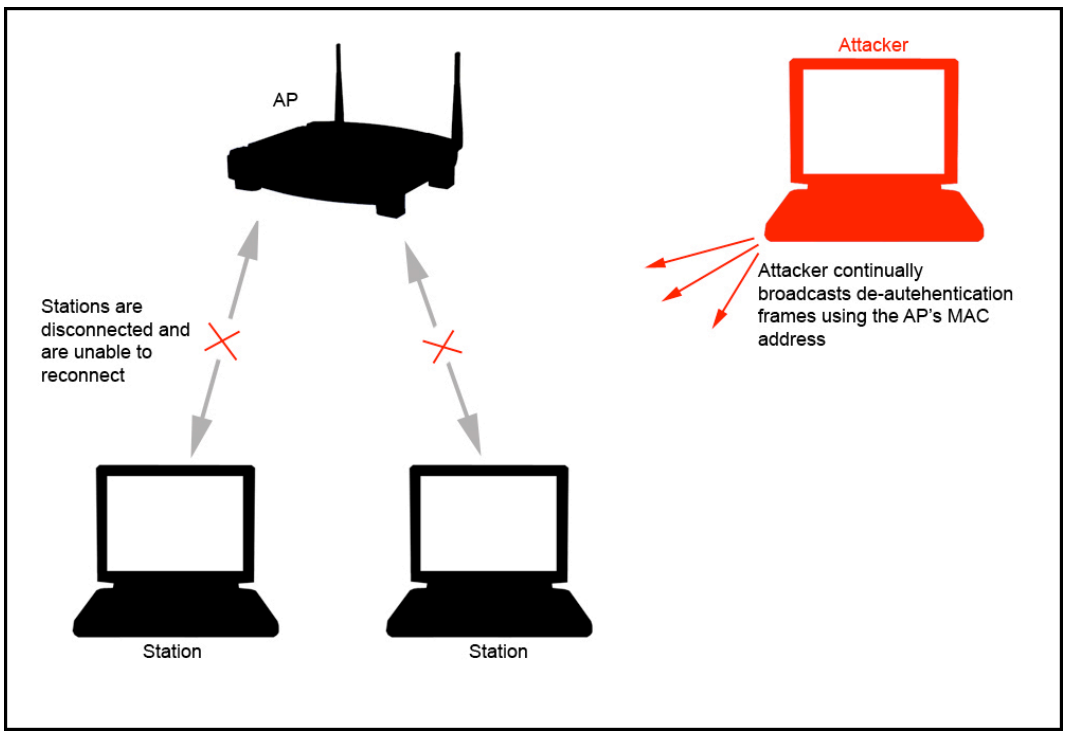


Figure 11 - A DoS Attack

A denial-of-service (DoS) attack is an attempt to make a computer resource unavailable to its intended users. Although the motives for carrying out such an attack can vary, it generally consists of the malicious effort of an individual to prevent a connectable service from functioning efficiently or at all.

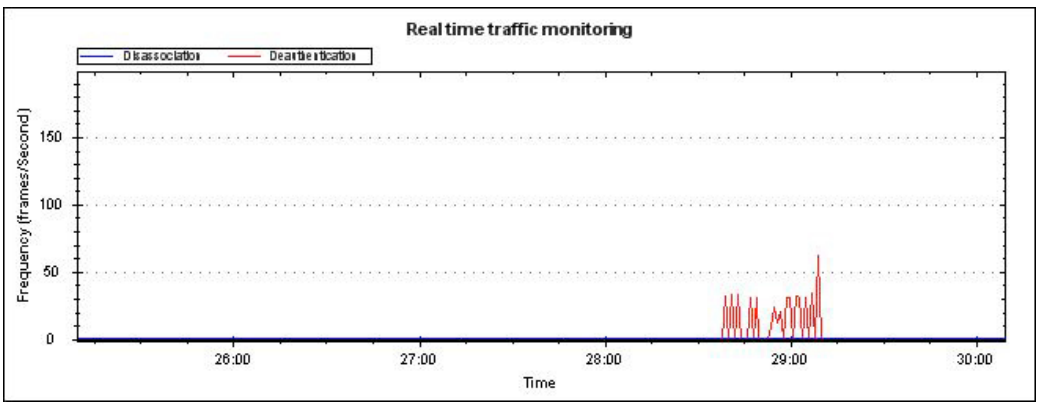
In the case of an 802.11 wireless network, DoS attacks can take place in a number of different ways such as a physical layer attack by causing interference to the medium (air) and also by spoofing (imitating) an Access Point and transmitting at a higher power to draw users into what is commonly referred to as a “black hole”.

In the case of this project, the DoS attack that will be investigated will essentially exploit the 802.11 state machine. As was discussed in section 4.1.3, this state machine dictates the different stages that take place when a station is connecting to an access point. If we consider a disconnected station to be both unauthenticated and unassociated then this can be considered as state one. State two sees the station moving to an authenticated state but still remaining unassociated. State three then sees the station both authenticated and associated and this is a fully connected state. The station/AP use a combination of management frames to carry out this connection (authentication packets and association packets) and move from state to state. When connected, in order to disconnect and return to state one, the station is sent another kind of management frame from the AP called a deauthentication packet. This returns the station to state one and it is disconnected.

Essentially, the DoS attack investigated here utilises this state machine as a basis for the attack. The attacker will simply flood the network with deauthentication frames from a specific Access Point, thus causing all (or a single) client to be disconnected and returned to state one. The attacker is in this case essentially spoofing the AP by sending management frames as if they are the AP. If the deauthentication packets are flooded continuously then the client should not have any success in re-connecting.

This attack is carried out by utilising the Aireplay-ng tool which is part of the Aircrack-ng suite detailed above. This tool will form deauthentication packets and transmit these across a chosen network. The tool is fed an AP MAC address and it will constantly transmit deauthentication packets to all clients connected to that AP. A single client can be specified though, and packets will only be sent to them.

This attack is very successful and as the characteristics of a DoS attack require, it fully disrupts the service to the network. The attack was performed and monitored with additional network monitoring software and the results from this software quite clearly demonstrate the effectiveness of the attack.



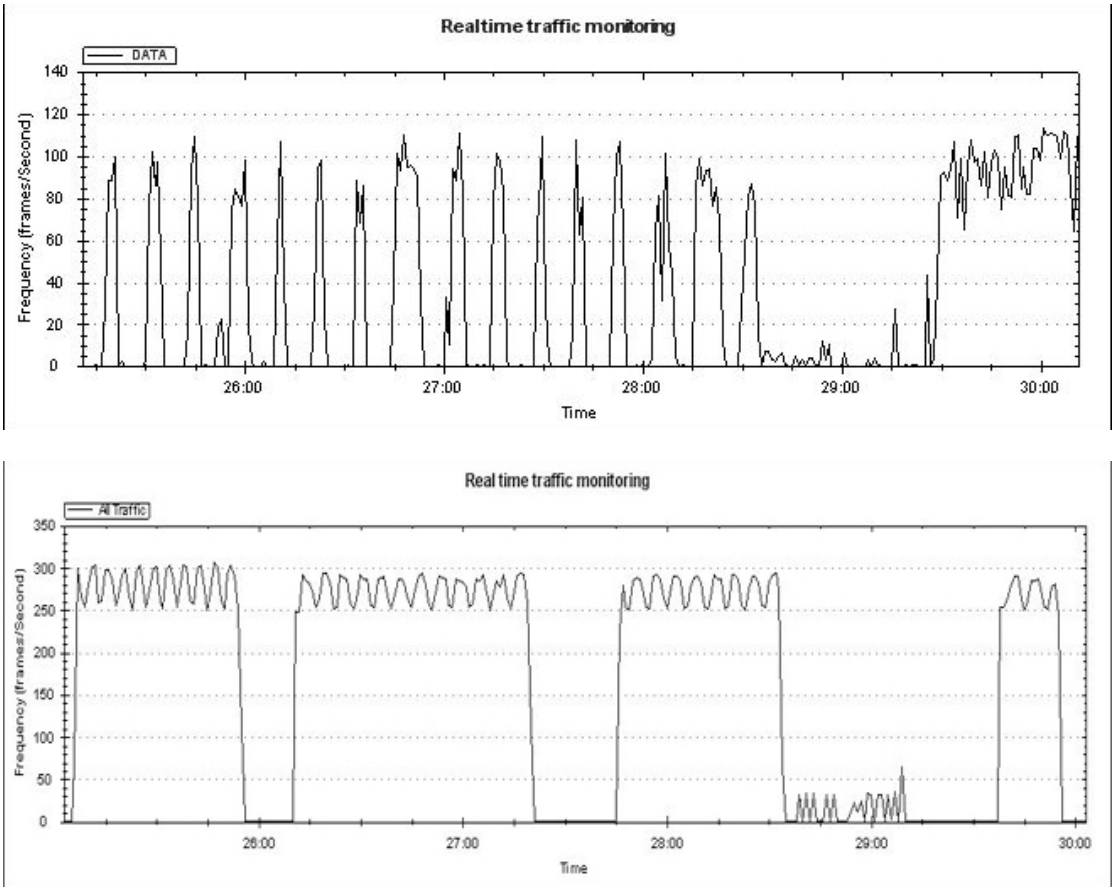


Figure 12 - Network Monitoring of a DoS Attack

This clearly demonstrates that network traffic, especially data traffic, was brought to a standstill for the duration of the attack, i.e. the period in red where deauthentication packets were being transmitted.

7.2 Cracking WEP

Another attack that was simulated was the utilisation of a software tool to crack the security applied to 802.11 wireless networks, initially looking at the commonly used WEP protocol.

Wired Equivalent Privacy (WEP) is an algorithm used to secure 802.11 wireless networks and it was first introduced in 1997. Since then, it has been depreciated in 2004 for use as a wireless privacy mechanism due to certain flaws in its security. While this is the case, many home and open networks still utilise this standard for basic encryption and security.

WEP utilises stream cipher RC4 for confidentiality. RC4 is a very popular cipher and it operates by creating a pseudorandom stream of bits based on a key. In 2001, three individuals, Scott Fluhrer, Itsik Mantin and Adi Shamir, managed to exploit the way that the WEP uses the RC4 cipher and it's Initialisation Vector (IV) in order to obtain the key used in a WEP encrypted session. 24-bits of the RC4 key make up the IV.

Firstly, let's look at this exploitation. An IV is a block of bits that is utilised to permit a stream cipher to produce a unique stream that is independent from other streams produced by the same encryption key and this is used to prevent repetition. As mentioned, WEP uses a 24-bit IV which is too small and means that there is roughly a 50% chance of the same IV appearing every 5000 packets, which on a busy network, isn't all that many. Fluhrer, Mantin and Shamir discovered that a flaw in the encryption algorithm caused certain IVs to reveal more information about the WEP key. [9]

By capturing a large amount of packets from a network, statistical analysis enables this key to be discovered. The key is essentially guessed using the information obtained from the weak IVs and the possibility that a certain byte in the key is correctly guessed goes up when a weak IV is captured. Thus, as more packets are captured, more keys are guessed and the most likely key will present itself.

The steps in this attack are carried out by utilising a range of tools from the Aircrack-ng suite that was explained in an above section and the diagram below shows the WEP attack.

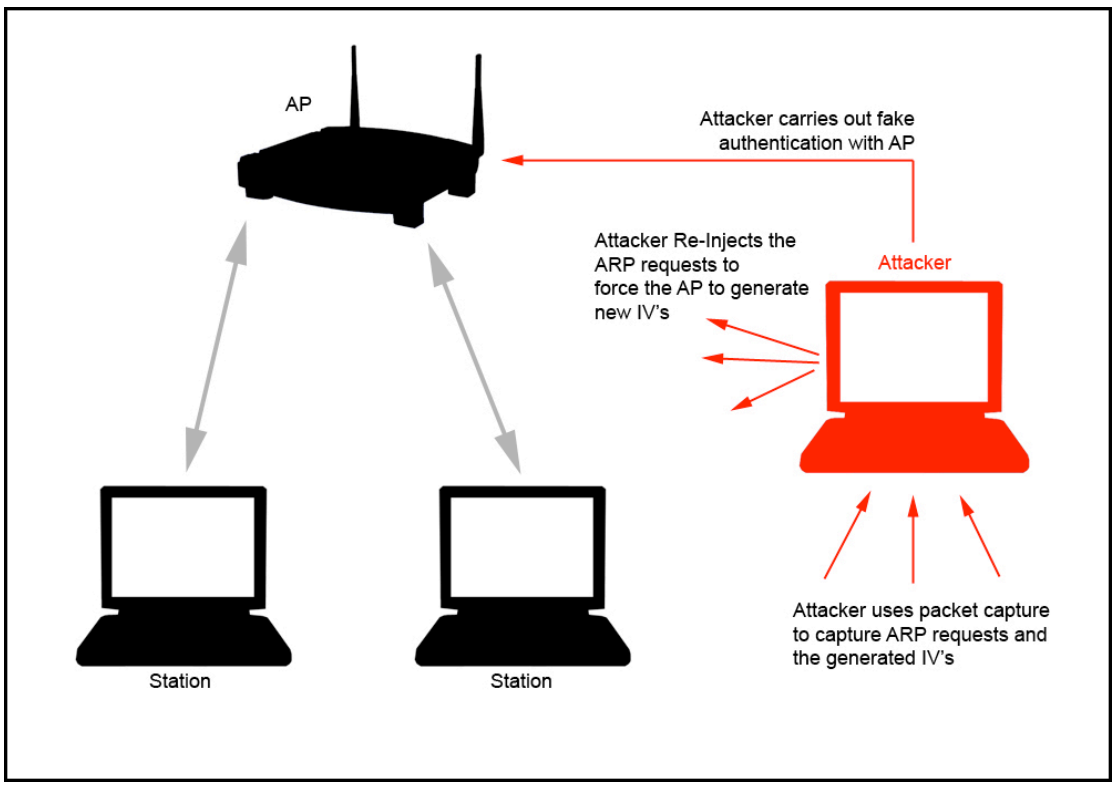


Figure 13 - A WEP Cracking Attack

Initially, a tool called Airodump-ng is utilised. As mentioned, this is used for packet capturing raw 802.11 frames and it will run in the background so that when the main part of the attack takes place, it can capture the IV's that are required to obtain the WEP key.

Then the attacker must carry out a fake authentication with the AP in question. The MAC address that the attacker will later re-inject packets from must be associated with the AP because if not, the AP will simply ignore any injected packets and as

such, will not generate any new IV's, therefore preventing the attack. The fake authentication can be carried out by using the Aireplay-ng tool.

Next the Aireplay-ng tool is utilised again except this time in ARP request reply mode. It will listen for ARP requests and will then immediately re-inject them back into the network. The reason that ARP request packets are selected is because the AP will almost always rebroadcast them and thus, create a new IV. The objective here is to obtain as many IVs as possible in a short time so this will deem quite effective.

As the new IVs are created, Airodump-ng will be capturing them and saving to an output file. When a sufficient amount of IVs have been obtained, this file can simply be used with the main Aircrack-ng tool which will implement the exploitation discussed above and yield the correct key.

The actual calculation of the key is an offline attack as the captured data is simply used with the Aircrack-ng tool but the steps leading up to that point involve significant packet injection and capture.

7.3 Cracking WPA

As WEP was deemed to be insufficient due to its easily exploitable security, WPA was proposed as a replacement to address these flaws.

Essentially, when WPA is deployed on a network, the traffic is encrypted using a 256 bit key. This can be entered as hex or as an ASCII passphrase, upon which a hash is performed to create the key. This is known as Pre Shared Key (PSK) mode

WPA is inherently secure and as such, only really susceptible to "Brute Force" attacks upon the network key. This means that if a simple, English language passphrase is used, a "brute force", attack may be successful. A "brute force" attack utilises a file that is filled with thousands (maybe even millions) of ASCII phrases and this is known as a "dictionary" file. The attack is named "brute force" because it essentially tries every one of the contained phrases until it finds a correct match. This can be quite effective but in essence, if even a 13 character, random ASCII phrase is used, this presents a large number of possible passphrases and it could take one computer years to obtain the key by brute force.

Thus in order to demonstrate this attack, a trivial example will be used by utilising a dictionary file that is known to have the English language passphrase in use contained within it. The diagram below illustrates the attack.

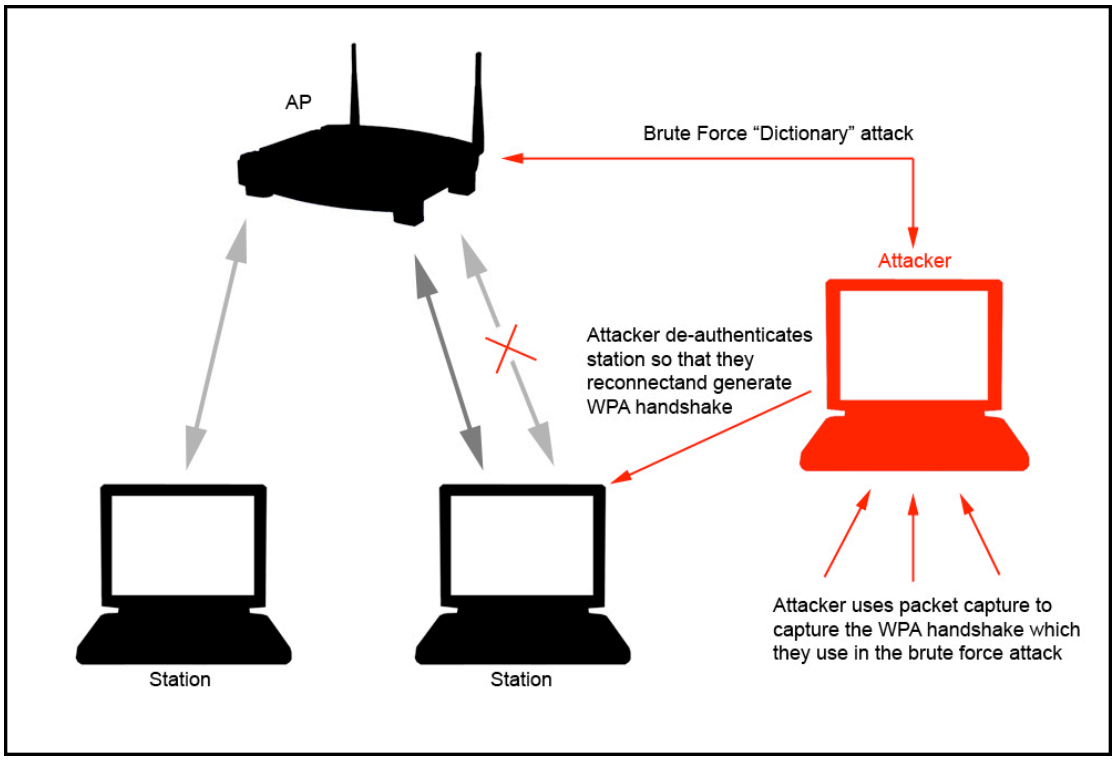


Figure 14 - A WPA Cracking Attack

To begin the attack, the Aircrack-ng suite of tools is utilised once again. Airodump-ng is started to enable the four-way handshake to be captured. This is a handshake that takes place when the AP is authenticating itself to the station. This handshake is essentially a means of exchanging the session master key without actually revealing it. Airodump-ng captures this handshake to a separate file so that it can be used when attempting to brute force the key.

In obtaining the four-way handshake, it will most likely be necessary to deauthenticate a client from the network in order to cause them to connect again and thus generate the handshake. This can be done by sending a smaller number of deauthentication packets, as in the DoS attack, to a single client.

Once the handshake has been obtained, the main Aircrack-ng tool is utilised with a dictionary and the file containing the handshake to perform the brute force attack. As mentioned, in this case, the passphrase is known to exist in the dictionary and thus the passphrase is found almost immediately as a short dictionary was used.

The computer attempts the brute force attack at roughly 60 passphrases per second so it is easy to imagine how long it would take to cycle through a few million passphrases; roughly 9 hours.

7.4 Rogue Access Point/Evil Twin

The Rogue Access Point/Evil Twin attack is essentially more of a way of instigating a number of attacks rather than being a specific attack in itself.

The main cusp of the Rouge AP is to create a virtual access point on an attacking machine that mirrors a trusted access point known by a victim. The idea is then to encourage the victim to connect to the rogue access point as opposed to the trusted one.

This process can form the beginning of a number of attacks, namely the black hole style DoS attack, a man in the middle attack or a phishing attack where the user thinks they are connected to a legitimate web service that is in fact a sponge for their critical information.

As mentioned, the Aircrack-ng suite contains a tool called Airbase-ng that can essentially carry out the task of creating a virtual access point and this can be successfully used to implement the attacks mentioned.

The use of a rogue access point is explored somewhat in the following section regarding the man in the middle attack.

7.5 Man in the Middle

The man-in-the-middle attack (MITM) is a form of active eavesdropping in which the attacker makes independent connections with a victim and its gateway. The attacker will relay messages between them, making them believe that they are communicating directly with each other over a private connection when in fact the data is passing right through the attacker.

In a MITM attack, attackers can simply intercept the data being transferred or they can both intercept and modify the data before passing it on. In the remit of this project the attempt will be to simply intercept and understand some pieces of data. The man in the middle attack will also be attempted in two different ways that are slightly different but both exist in the data link layer.

7.5.1 Using Rogue AP

The first way that this attack will be carried out is by utilising the rogue access point attack discussed above. This will create a virtual AP as has been mentioned and this can then be used to encourage the victim to connect illegitimately. This connection with the victim can then be bridged with a connection to the legitimate AP and the victim's traffic forwarded through while being observed.

The Airbase-ng tool is used for this attack to create a fake AP. The user can either set the name of this AP or allow the AP to set its own name based on the Probe requests that it intercepts from connecting stations.

Essentially, as the attacker utilises the MadWiFi driver, they are able to create a number of virtual AP's (VAPs) on the one Network Interface Device. As normal, the attacker will utilise one of these VAPs to form a standard connection with an AP (or any other legitimate connection will suffice). They will additionally create another VAP that will be used by Airbase-ng to create the rogue access point on. The next stage is to successfully bridge these network connections and enable IP packet forwarding between them. This is one of the most important tasks as, when the victim connects to the fake AP, they must still be able to gain seamless access to the internet for

example. By bridging the attackers legitimate connection with their fake connection, anyone who connects to the fake AP will essentially be using the other legitimate connection. The main difference is that their traffic will be flowing through the attacker and they will be able to view it, or in some cases, even modify it.

In order for this attack to be successful, the attacker must also run a DHCP server locally. This ensures that when a victim connects to the fake AP, they are given a legitimate IP address. This provides both the illusion of legitimacy as well as a real connection to the attacker.

The diagram below shows the MITM attack using a rogue access point.

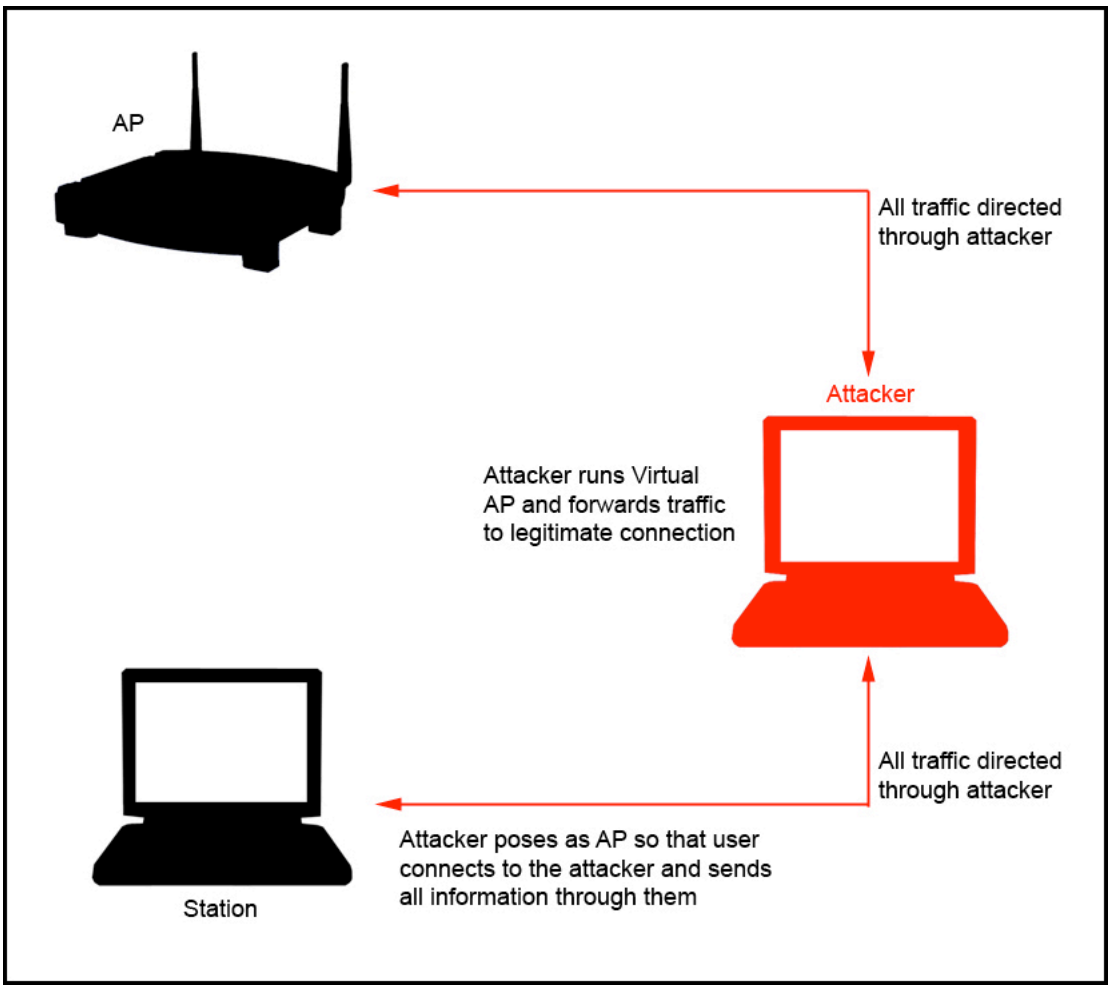


Figure 15 - MITM Attack using a Rogue AP

Unfortunately, the downside to this attack is that it is extremely complicated and relies on a significant number of different tools working together in order to attack successfully. Due to its complexity, this attack was never successfully implemented in the laboratory and thus could not be tested properly. Numerous problems arose with ensuring IP forwarding worked between the two connections properly and successfully implementing the DHCP server was a tricky task. This being said, the concept of the attack is well understood through numerous and increased attempts to implement it.

7.5.2 ARP Poisoning

This attack can also be carried out in a different manner by using a tactic known as ARP poisoning. The Address Resolution Protocol (ARP) is the method for finding a host's link layer address, i.e. MAC address. This attack basically allows the victim to remain associated with the legitimate access point but to send all its data through the attacking machine and vice versa.

The diagram below displays the MITM attack using ARP poisoning.

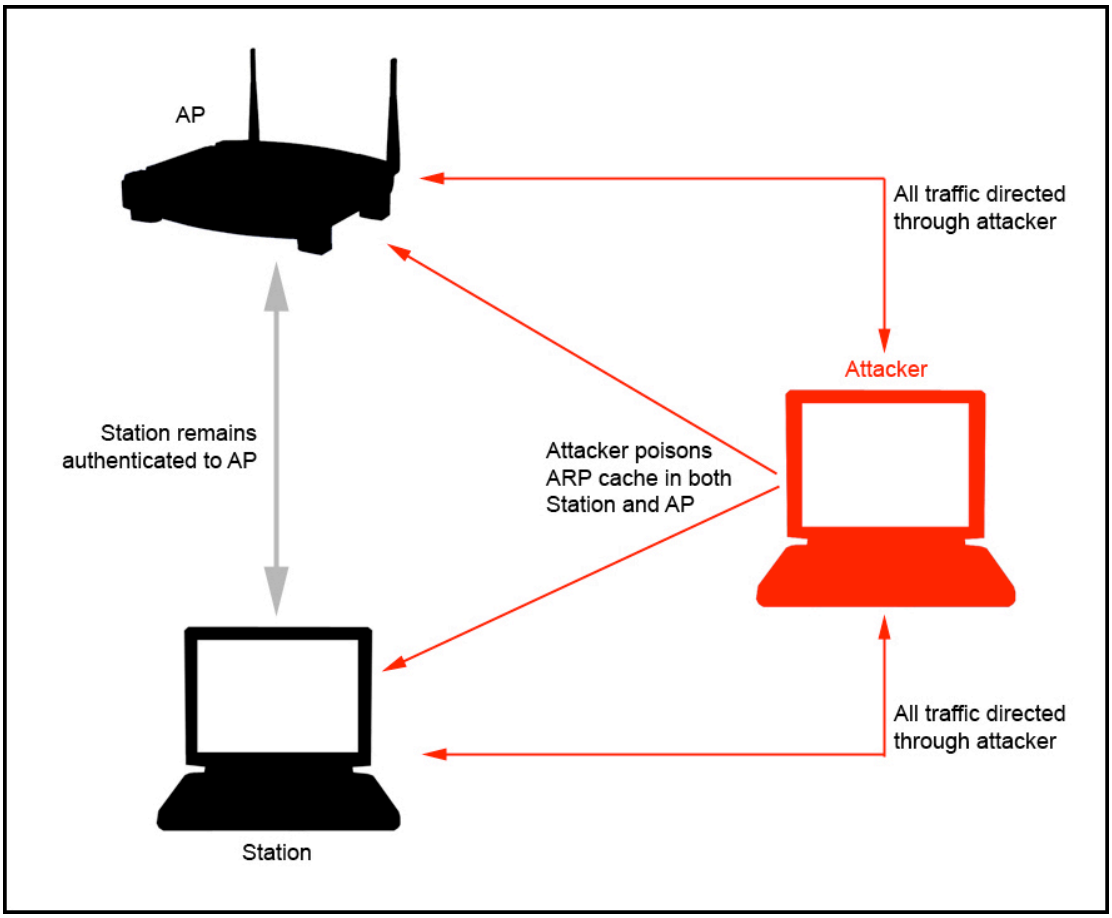


Figure 16 - MITM Attack using ARP Poisoning

The principle of ARP poisoning is to send fake ARP messages to the victim machine (and vice versa to the AP) which will include the MAC address of the attacking machine and thus “poison” the ARP cache in the victim machine (and AP) so that they will send packets to the attacking machine.

This alone would be sufficient to cause a significant DoS attack, but in order to continue with the MITM attack, the attacking machine needs to ensure that IP forwarding is enabled within its kernel so that the packets arriving destined to (and coming from) the AP can be forwarded on. This means that essentially, as the name suggests, the attacker has become the “man in the middle” of the connection.

Now, if the attacker utilises a remote browser plug-in, supplied by a tool such as Ettercap, the users web traffic, POP3 details, FTP passwords, forum logins and instant messenger details can be obtained and viewed on the attacker screen.

This attack is extremely effective at obtaining the above-mentioned data and for general breaches of privacy and it is essentially completely transparent to the user.

7.6 Current Methods of Detecting 802.11 Attacks

Certain measures are employed in different 802.11 situations in order to attempt to detect these attacks taking place and they vary from situation to situation.

One such employed method is to use Intrusion Detection Systems (IDS) to attempt to detect such activity. An IDS can be used to detect several types of malicious behaviours on a network that can compromise security and trust. Mostly, they examine and analyse the network traffic and the patterns that it takes. This way, certain malevolent patterns can be flagged and a warning issued or action taken. Two popularly used pieces of software are Snort and Kismet. Snort performs packet logging and real time traffic analysis. It attempts to actively block or passively detect a variety of attacks by performing protocol analysis and content searching/matching. Kismet is a passive network detector and IDS that uses drones to capture packets and log them with a server to perform analysis of the traffic and attempt to detect malicious behaviour and attacks.

There also exists certain commercial deployments such as was discussed in section 3.3 when discussing the Cisco Unified Wireless LAN. This deployment for example uses an array of air monitors deployed across the entire WLAN deployment which are used to passively sniff traffic. The use of multiple sniffers enables this kind of system to determine the position of the APs in the deployment and ensure that they are detected in the correct place. Systems like these often deploy additional hash checks on all frames sent/received and they are able to do this as they know exactly who will be using the system, i.e. their employees, and the hardware they will be using. Deployments like this are always very specific to the area they are deployed in.

8 Development of Packet Capture Software Tool

As mentioned when discussing trust systems in general, once the threat model is understood it is important to implement a mechanism for collecting information about the system so that it can be classified in a certain way and used as a metric for determining trust.

Therefore, as stated in the objectives, the next main task of the project is to develop a software tool that will perform packet capture in order to record information from the packets in the air.

This section of the report will outline the basic specification and design process for a packet capture tool, giving an overview of the development stages of the software and outlining the programming methods used to implement the tool. There will also be significant discussion into the metrics calculated and used and how these are refined to ensure the use of the most viable data.

As a note, code snippets will be supplied within this document where it is deemed necessary but a full listing of the commented program code can be found in Appendix 1.

8.1 Specification of Tool

The tool that is developed will be a simple packet capture tool and it will utilise the libpcap library with the C programming language in a Linux environment. This API was discussed in the tools section previously and as mentioned, it simply provides a number of functions that can be used alongside the correct hardware to implement packet capture.

8.1.1 Hardware

In order to utilise the libpcap API, the ability must exist to place the 802.11 Network Interface Device that is to be used for the packet capture into monitor mode. With common on board 802.11 hardware, such as the popular Intel devices, updating drivers to enabling monitoring can be both tricky and provide intermittent functionality. For this reason, separate hardware will be used, in this case the Linksys WPC55AG wireless adapter which contains the Atheros AR5001X chipset. As mentioned before, this means that the MadWiFi driver will be in use and thus, the network adapter can be easily placed into monitor mode for use by libpcap.

A simple way of placing the device in monitor mode is to utilise a small tool provided by the Aircrack-ng suite called Airmon-ng. This simply takes the MadWiFi parent device as a parameter and creates a VAP for the device that operates in monitor mode.

8.1.2 Utilising Radiotap

In order to fully specify the packet capture tool it is important to decide what data is going to be captured from the packets themselves in order to develop metrics that can determine trust. As mentioned previously the MadWiFi driver utilises the Radiotap standard for appending additional information to the frame.

It is proposed that certain elements of this Radiotap information be utilised in order to attempt to build a number of trust metrics. The Radiotap information that will be used is the dB Antenna Signal and the Packet Rate.

The dB Antenna Signal field of the Radiotap header provides information on the RF signal power at the antenna and essentially provides a read out of the received signal strength of the frame. If the attacks detailed above are considered, the attacker almost always injects or broadcasts frames as if they are the Access Point, i.e. they "spoof" the AP to the users. If the user was to monitor certain metrics involving the power level, then the thinking is that when an attack takes place, these metrics should provide information to indicate it. It is also important to note that no trust determination tool in use in the field today utilises power as a metric.

Similarly with the Packet Rate; when a station is communicating with a legitimate AP, the packet rate should be relatively constant. Introduce an attack and this might cause some fluctuations that alongside additional metrics could provide more information regarding an attack taking place.

8.1.3 Development Specification

The following is a general specification for the development of the software tool.

- Open a monitoring device
- Implement a packet capture loop with this device
- Determine whether captured frame is from an Access Point or from a Station
- Gather data from the frame to use for metrics
- Store this data relative to each AP and Station
- Carry out calculations on gathered data to determine useful metrics
- Store information for future analysis and processing
- Display information to user

8.2 Initial Development Phase

The initial development of the software tool will be mainly focused on implementing the simple packet sniffing operation and focusing on parsing the raw data retrieved. In this phase, no attention is given to implementing storage or data structures to maintain the data as this can be developed after the main aspect of the software is understood and put in place.

8.2.1 Initial Structure

The initial structure of the software will be as follows

- Use libpcap to open a monitoring device that can be used for packet capture
- Create a thread
- Run the libpcap capture loop within this thread
- Read in packets and split out the Radiotap Data and the MAC Data
- Parse the specified Radiotap data and Frame Control Section of the Frame
- Print MAC address of frame and parsed data to the User

8.2.2 Opening a Monitoring Device

In order to begin creating a packet sniffer, the interface that is being used to capture on must be determined and enabled. In the case of this packet capture tool, as mentioned, the Atheros driver is in use and a monitoring device *ath1* is created. This device name is then passed as a string to the libpcap initialisation function, *pcap_open_live( ),* to open the device and initialise the sniffer. This can be seen in the code fragment below. It is important to note that this monitoring device will perform channel hopping so as to capture packets from all AP's in use.

```
G.handle = pcap_open_live("ath1", 2048, 1, 1000, errbuf);
if (G.handle == NULL)
{
      fprintf(stderr, "Couldn't open device: %s\n", errbuf);
      return(2);
}
```

As can be seen, a handle is used to differentiate between devices. If so desired, multiple devices could be used to sniff on by using multiple handles. This is just like opening a file for reading or writing, the sniffing "session" must be named uniquely.

As a short note, variables prefixed with *G.* are global variables contained in a separate global structure that resides in a simple header file for variables. This allows better management of global variables.

Looking at the *pcap_open_live* function call, there are a number of parameters passed to it. The *ath1* parameter is simply the device as discussed and the rest of the code fragment tells that device to read a maximum of 2048 bytes, to operate in

promiscuous mode, to sniff until an error occurs, and if there is an error, store it in the string *errbuf*.

The program then immediately checks to see if the device has been opened accordingly by checking to see if the handle is null or not. If it is, the *errbuf* error message is printed to the screen and the program exits.

As mentioned, the device is opened in promiscuous mode and what this basically means is that the device will sniff all traffic in the medium, not just traffic that directly related to it. The device will also channel hop to find all APs/STAs in its vicinity.

8.2.3 POSIX Threads

In order for the program to operate efficiently and give realistic results, it is important for it to carry out sniffing while continuously updating metrics alongside this sniffing. In order to achieve this, the program must utilise threads. This can be quite a difficult concept to grasp and implement and therefore it is important to understand threads fully before implementing them.

Within UNIX systems, a standardised C threads programming interface is specified and this is known as POSIX Threads, otherwise referred to as pthreads. These are used in a C program by simply including a *pthread.h* file and then by utilising the functions provided by the API.

This aside though, what is a thread? Essentially, a thread is an independent stream of instructions that can be scheduled to run by an operating system. The best way to think of this is almost as if a thread is a separate procedure that runs independently from the main program. A “multi-threaded” program would simply be a program that contained a number of procedures that are all able to run simultaneously.

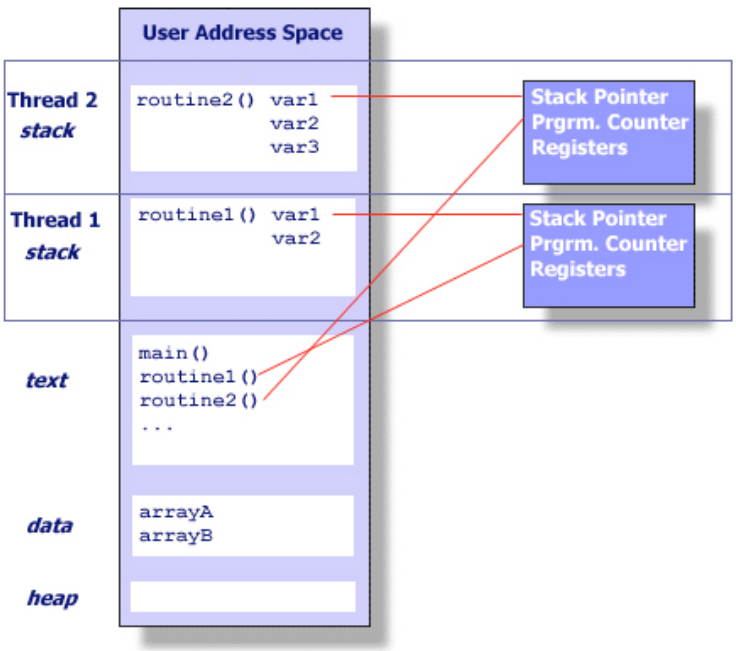


Figure 17 - The user address space for threads

The above image displays a number of threads running within a UNIX process. Essentially, each thread uses that process's resources and it has its own independent flow of control which is accomplished because a thread maintains its own stack pointer and its own registers. A thread is considered to be very "lightweight" because most of the overhead has already been accomplished through the creation of its process.

Therefore, as the packet capture program in question is required to operate a number of tasks in tandem, i.e. sniffing and metric calculation, then threads will be required so that these tasks can run simultaneously.

POSIX threads also include the use of mutex's which stand for MUTual Exclusion and are used to lock data structures when being accessed by a thread.

8.2.4 pcap Loop

In order to begin capturing packets once the device is initialised, the packet capture loop needs to be initiated and this is carried out by utilising the *pcap_loop( )* function. Before this function can be fully detailed, the concept of a callback function needs to be explained.

In programming, a callback is a piece of executable code that is essentially passed as an argument to other code within the program. Let's consider a program that is waiting on an event, for example, listening for a button press and every time that button is pressed a function is called. This is known as a callback function.

Callback functions are used in libpcap in the aforementioned packet capture loop. Essentially, the *pcap_loop( )*function instigates a loop and whenever a packet is sniffed, then the callback function (provided as a parameter to the *pcap_loop( )* function call) is called.

Below is a snippet from the program instigating the *pcap_loop( )*.

```
main()
{
/*....rest of main function....*/

pthread_t thread1;
pthread_create(&thread1, NULL, start_loop, NULL);
}

void *start_loop()
{
      pcap_loop(G.handle, -1, process_packet, NULL);
      pcap_close(G.handle);
}
```

As can be seen, the thread is created by calling the *pthread_create( )* function with four parameters. The first parameter is a unique identifier for the thread itself, created just previously as a standard variable declaration. The second parameter defines the attributes that the thread has and in this case it is presented as NULL, meaning that

the function will utilise the default values. Next, the routine that the thread will execute is passed as a parameter, in this case *start_loop* which is seen below the main function and this contains the important *pcap_loop( )* function call.

As can be seen, the *pcap_loop( )* call takes a number of parameters also. The first is the same libpcap handle that was created previously when opening the monitoring device for sniffing. The second parameter states the number of packets that are to be sniffed and if this is passed as a negative number, as is the case with this program, then sniffing happens continuously until an error occurs. The third parameter specifies the name of the callback function that is called when a packet is sniffed, in this case it is a function called *process_packet( ).* The final parameter provides a means to pass user generated arguments to the callback function itself and as there are none in this program, this is passed as NULL.

8.2.5 The Callback Function

Now that the *pcap_loop( )* function call has been defined, it is important to discuss the format of the callback function, which in the above code snippet was seen to be *process_packet( )*. This function cannot be arbitrarily defined as the *pcap_loop( )* needs to know how to use it. The general prototype for the callback function is seen below.

*void got_packet (u_char *args, const struct pcap_pkthdr *header,*
*const u_char *packet);*

The first argument here corresponds to the final argument passed to *pcap_loop( )* which was NULL so this parameter can be given any arbitrary name as it will not be used. The second argument is the pcap header which is specified in the *pcap.h* file that is included with this program. This contains data regarding the timestamp of the packet capture and the packet size. None of this data is used throughout the program in question but this must be included in the format of the function. Finally, the last parameter is a u_char pointer which is basically the raw bytes of the packet itself that can be used.

Below is a code sample showing the intial *process_packet( )* function.

```
void process_packet(u_char *whatever, const struct pcap_pkthdr
                    *pkthdr, const u_char* packet)
{
      count++;
      printf("%d\n", count);

      struct ieee80211_radiotap_header *radiotap;
      radiotap = (struct ieee80211_radiotap_header*) packet;
      const struct header *frame;

      frame = (struct header*) (packet+radiotap->it_len);

      printf("%02x:%02x:%02x:%02x:%02x:%02x --> ", frame-
      >address2[0],frame->address2[1],frame->address2[2],frame-
      >address2[3],frame->address2[4],frame->address2[5]);

      process_rtap(radiotap);
      printf("Power: %d  Packet Rate: %d", G.pwr,
      G.pkt_rate_100kHz);
}
```

Firstly, the program creates a pointer to an *ieee80211_radiotap_header* structure called *radiotap* and points this at the beginning of the packet data where the Radiotap header resides. This pointer can now be used for parsing that radiotap data. This structure is defined in *ieee80211_radiotap.h* and is seen below.

```
struct ieee80211_radiotap_header {

      uint8_t     it_version;
      uint8_t     it_pad;
      uint16_t    it_len;
      uint32_t    it_present;

};
```

After this, a pointer is then created to the *header* structure that is contained in the *802_11.h* file and this is called *frame*. This is essentially a structure to represent the MAC header and as can be seen in the code snippet, this is pointed at the packet data with the *ieee80211_radiotap_header* structure field *it_len* added to it. This field represents the length of the radiotap header, so therefore the radiotap data is jumped and *frame* points at the start of the MAC data.

The *header* structure is defined in *802_11.h* and as mentioned, it represents the MAC header that was discussed previously in section 4.1.2. It can be seen below.

```
struct header{

      u_char frame_ctl[2];
      u_char duration_id[2];
      u_char address1[6];
      u_char address2[6];
      u_char address3[6];
      u_char seq_ctl[2];
      u_char address4[6];

};
```

The remainder of the callback function simply prints the *address2* element of the *header* structure in hex which, as was discussed, is the senders address. It then passes the pointer to the radiotap data to a function called *process_rtap( )* which retrieves the received power and packet rate data to global variables *G.pwr* and *G.pkt_rate_100kHz* which are then printed to the screen beside the MAC address of the frame.

This is not an entirely useful output as it simply outputs a printed line for each frame that is received. For the sake of initial development though, this is suffice and will be developed upon later.

8.2.6 Parsing the Data

In order to capture the desired radiotap data in the above code, a pointer to the beginning of the radiotap header is passed to a function called *process_rtap( ).* As mentioned, this reads and interprets the radiotap data and sets certain global variables but it is important to explore how this takes place and this is not exactly a straightforward task.

If the diagram in Figure 10 is considered when the radiotap specification was detailed, the most important thing to notice is that the section of the header that contains the data is of a variable length and content. This, of course, is because the header can be flexible in the radiotap fields that it contains. Thus, the header cannot just be chopped off the frame and placed into a set structure as there is no way of knowing what fields are present. Instead, an iterator must be used. This is essentially a piece of code that jumps through the fields that the header does contain and uses a switch statement at each jump to check if the current field is one of the desired fields.

In order to save time and remove the extremely complex task of coding this radiotap iterator from scratch, a version of the radiotap iterator already exists and is included in the wireless source directory most modern Linux distributions and also in the source code of many popular open source wireless packet capture tools. It is provided as a group of functions and is contained within *radiotap.c* and *radiotap.h.* [10]

The following code snippet shows that *process_rtap( )* function that utilises the iterator.

```
void process_rtap(struct ieee80211_radiotap_header* rtap_header)
{
      G.pkt_rate_100kHz = 0, G.pwr = 0;
      struct ieee80211_radiotap_iterator iterator;
      int ret = ieee80211_radiotap_iterator_init(&iterator,
      rtap_header, rtap_header->it_len);

      while (!ret) {

            ret = ieee80211_radiotap_iterator_next(&iterator);
            if (ret)
                 continue;

            switch (iterator.this_arg_index) {

                  case IEEE80211_RADIOTAP_RATE:
                  G.pkt_rate_100kHz = (*iterator.this_arg) * 5;
                        break;

                  case IEEE80211_RADIOTAP_DB_ANTSIGNAL:
                  G.pwr = (*iterator.this_arg);
                        break;

                  default:
                        break;
               }
        }
}
```

As can be seen, the function takes a pointer to an *ieee80211_radiotap_header* structure as an argument. The function immediately initialises the global variables for power and packet rate to zero and then creates a new *ieee80211_radiotap_iterator* variable called *iterator.* An integer that is used for controlling the iteration loop is declared as *ret* and then set equal to the initialisation of the iterator which is done by calling the *ieee80211_radiotap_iterator_init( )* function. This function takes three arguments; the iterator itself, the pointer to the radiotap header and the length of the radiotap header.

A while loop is then entered and the iterator is set to the next element in the header by calling the *ieee80211_radiotap_iterator_next* function and passing *iterator* as a parameter. This then sets a new value for *ret* so that the while loop can be exited when the iterator reaches the end of the header.

A switch statement is then carried out on *iterator.this_arg_index* which is essentially an integer index that represents the fields from the header that the iterator is currently looking at. The cases in the switch are then set to be *IEEE80211_RADIOTAP_RATE* and *IEEE80211_RADIOTAP_ DB_ANTSIGNAL* which are defined in an enumeration in the *ieee80211_radiotap.c* file. As the iterator cycles through the fields, the switch will extract the desired fields when the index matches the case and assign the power and packet rate variables by setting them equal to *iterator.this_arg.*

The use of this iterator simplifies this task greatly and enables the radiotap header to be cycled through until completion, with the switch checking for fields at each iteration and extracting the data accordingly.

8.3 Further Development

The previous section fully outlines the way in which the packets are captured and how the appropriate data is parsed from them. As was seen, this was simply printed to the screen on a per packet basis which in essence, is entirely un-helpful and provides no means of categorising an AP so that metrics can be calculated for it.

In order to ensure this software tool is actually useful and can provide functionality for determining trust metrics, there will need to be a mechanism in place to both record data, store it for calculation and do this all or a per AP/Station basis.

8.3.1 Proposed Functionality

The diagram below outlines the proposed structure for the further development of the software tool.

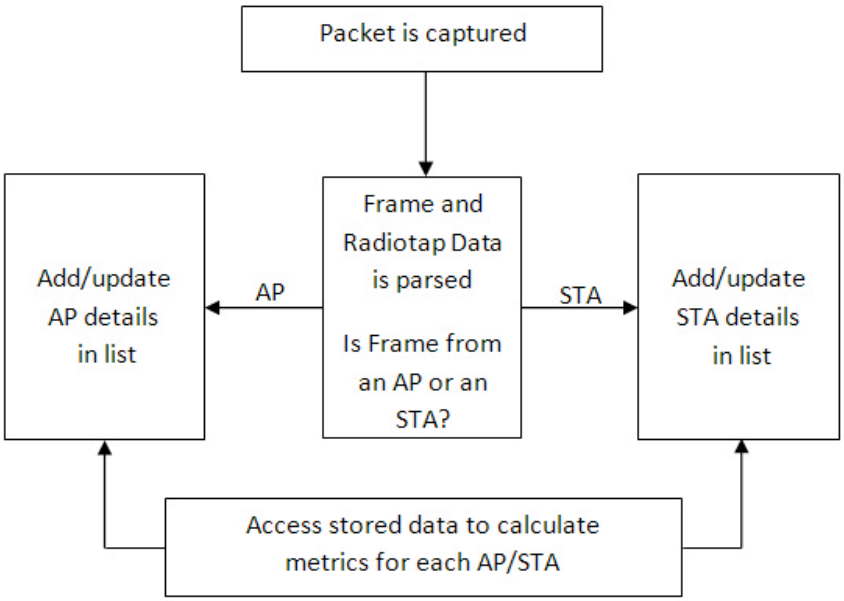


Figure 18 - Structure for further development

Essentially, when a packet is captured and the information from it parsed, the tool will determine whether the frame came from an AP or from a user station. The tool will also maintain two lists, one of the APs that it sniffs and one of the Stations. When it has been determined where the captured packet came from, then that AP/Station will be added to the appropriate list, or if it is already in the list, its data updated from the newly captured packet. Thus, over time, the tool will build up a list of the APs and stations that are in its vicinity.

With these lists in place, the tool can then calculate metrics based on the data gathered. The tool will do this by calculating metrics for each five second window of data capture for each AP/STA.

8.3.2 Implementing Linked Lists

As mentioned, each AP/STA "seen" by the tool will be stored in an appropriate list and the best way to implement this is to use two linked lists, one for APs the other for STAs.

A linked list is essentially a data structure that consists of a sequence of data records (in this case AP or STA structs) such that each record has a reference, or a link, to the next record in the sequence. When the sequence reaches the end, then the final record simply has a link to NULL.

This is a very simple and popular data structure that will work well for the task at hand. The packet capture tool will employ linked lists by utilising an open source linked list implementation that is provided by the Comprehensive C Archive Network (CCAN). [11] It basically defines two types; *struct list_head* which is used for anchoring a list, and *struct list_node* which is usually embedded in the structure that is placed in the list. The implementation also provides a number of functions that are useful for implementing certain functionality with the list that can become transparent to the developer.

The following code snippets show how the *struct list_head* and *struct list_node* are used.

```
struct apList{

      struct list_head        accessPoints;
      unsigned int            num_accessPoints;

};

struct accessPoint{

      /*AP Information
      ......*/
      struct list_node  apList;

};
```

Once the structures have been set up like this, then the list can be initialised and used. A global instance of the list struct (apList above for example) is declared and the head of this is passed to an initialisation function to set up the list as can be seen below.

```
list_head_init(&G.apList.accessPoints);

list_head_init(&G.staList.stations);
```

When an AP struct is created then, for example, it can be added to the end of the list as follows.

```
list_add_tail(&G.apList.accessPoints, &G.accessPoint->apList);

G.apList.num_accessPoints++;
```

The *list_add_tail* function is passed, as parameters, the head of the list and the *list_node* struct that sits within the AP struct for the AP to be added. This will add the given AP struct to the end of the list and this would be implemented indentically for STAs also.

If an AP or STA is already in the list, then the captured data needs to be added to the correct node of the list ascertaining to that AP/STA. I this case, the list will have to be looped through until the correct node is found and then the data can be updated.

This is done by using the following function which is passed the list head, the struct that contains the list node and the *list_node* member of the structure as parameters.

```
list_for_each(&G.apList.accessPoints, G.accessPoint, apList)
{
      .......
}
```

The CCAN list implementation is rather powerful and contains a considerable amount of functionality regarding these linked lists but for the scope of the packet sniffing tool in development, this will be all the functionality that is used.

8.3.3 Storing the Data and Classification of Frame

Before classification of the captured packet can take place, the structures to be stored in the linked list need to be determined. Essentially, as discussed, there will be a list of all the APs sniffed by the packet capture tool and their data for calculating trust will be stored in the node also. This will be the same for stations.

The following examples are displayed in terms of Access Points but are virtually the same as is implemented for stations.

The following is a structure for an AP. The *NO_PACKETS* definition is simply a large number for now so as to create enough room for the power and packet rate data to be stored.

```
struct accessPoint{

      u_char            mac[6];
      int               packet_rate[NO_PACKETS];
      int               pwr[NO_PACKETS];
      int               index;
      int               avgpower, varpower;
      double            sdpower;
      struct list_node  apList;

};
```

When a packet is captured by the tool, the *process_packet* function is carried out and the MAC frame and radiotap data are parsed as was outlined previously. The packet must then be classified and stored in the appropriate linked list based on whether it is an Access Point or not. This is done by using the function detailed below.

```
int isFrame_ap(const struct header *frame)
{
      if(frame->frame_ctl[1] & (0x40) == 0x40)
      {
            return 0;
      }
      else
      {
            return 1;
      }

}
```

This function is passed a pointer to the MAC frame and the *frame_ctl[1]* aspect of the MAC frame struct is inspected. This represents the first byte of the two-byte frame

control section of the MAC header. As was detailed when outlining the 802.11 background theory, the second bit of this byte is the FromDS field so therefore, if it is masked with 0x40 the value of this field can be determined and then we can return accordingly to state whether the packet is from an AP or from a station.

Using this function within in the *process_packet* function allows the classification of the packet before it is added to the appropriate list. The code sample below shows an AP being added to (or updated in) the list.

```
if(isFrame_ap(frame))
{

      if(!(inAPlist(frame->address2)))
      {
            pthread_mutex_lock( &G.mutex1 );
            G.accessPoint = malloc(sizeof(*G.accessPoint));

            memcpy(G.accessPoint->mac, frame->address2, 6);
            G.accessPoint->index = 0;
            G.accessPoint->pwr[G.accessPoint->index] = G.pwr;
            G.accessPoint->packet_rate[G.accessPoint->index] =
            G.pkt_rate_100kHz;


            list_add_tail(&G.apList.accessPoints, &G.accessPoint-
            >apList);

            G.apList.num_accessPoints++;

            pthread_mutex_unlock( &G.mutex1 );
      }
      else
      {
            pthread_mutex_lock( &G.mutex1 );
            list_for_each(&G.apList.accessPoints, G.accessPoint,
            apList)
            {
                  if(G.accessPoint->mac[4] == frame->address2[4] &&
                  G.accessPoint->mac[5] == frame->address2[5])
                  {

                        G.accessPoint->index++;
                        G.accessPoint->pwr[G.accessPoint->index] =
                        G.pwr;
                        G.accessPoint->packet_rate[G.accessPoint-
                        >index] = G.pkt_rate_100kHz;

                  if(G.accessPoint->index == NO_PACKETS)
                  {
                        G.accessPoint->index = 0;
                  }

                  }
            }
            pthread_mutex_unlock( &G.mutex1 );

      }
}
```

As can be seen, once it has been determined that the packet is from an access point, it then checks to see if the AP is already in the list. This is simply done by using a function *inAPlist( )* which cycles through the entire linked list using the *list_for_each( )* function and comparing the MAC address of the captured frame that is passed to it with the *mac[ ]* variable of each node struct. Note that it only compares the latter two elements of the MAC addresses. This is due to an unfathomable bug that seemed to corrupt the first part of the MAC address. This is not an issue as the last two elements of the MAC address is the mainly unique part of the address and as such it still represents the AP/STA appropriately.

If a match isn't found then a new AP is created by dynamically allocating memory to *G.accesspoint,* a global AP struct. The MAC address is then copied to this from the frame and the index value set to zero. This index value is used as both the current index value for the power and packet rate arrays and also as an indication of how many packets have been received for an AP. As this is currently zero, the intial elements of each array are populated with the parsed values of power and packet rate. The *list_add_tail* function is then used to add this node to the AP list.

On the other hand, if a match is found, the *list_for_each( )* function is used to cycle through the list and match the MAC address of the frame to an AP in the list. When this match takes place, the index is increased by one and the parsed values of power and packet rate are added to the list node.

It is important also to note the use of the functions *pthread_mutex_lock(&G.mutex1)* and *pthread_mutex_unlock(&G.mutex1)*. As mentioned, a thread mutex is used to lock data structures when being used by a thread so that another thread cannot try to access them at the same time. In the code sample above, each time the linked list is accessed, the thread mutex is locked and then when the list access is complete, the thread mutex is unlocked. This is used across the entire program when the data structures are accessed.

8.3.4 Data Used and Metric Calculation

Now that the lists of APs and Stations can be populated and the captured data stored based on the AP/STA that it came from, a mechanism must be developed to calculate metrics based on this data.

As mentioned, one of the design specifications for further development was that the software tool would calculate metrics for every 5 second window of captured data. This is an important point because this way, not only will the software tool record the metrics over time, updating at regular intervals, it will calculate realistic metrics for each AP/Station in that window as it will only be considering captured data from that 5 seconds in time.

It's also important to note at this stage of the development that packet rate will no longer be investigated towards developing a metric of trust and only the power values will be considered and this is for a number of reasons.

Firstly if the attacks detailed previously are considered, almost every attack involves the malicious user "spoofing" an AP and sending packets as if they are the access point. The main thinking here is that if an attacker gets involved in a network and spoofs an AP that the signal received from that AP will be of a different power level to

the legitimate access point. This enables the captured power value to be used extremely effectively to develop statistical metrics.

As for packet rate, this may be a factor to consider further down the line as a supplement to the power metrics but presently, by itself, it does not appear that the packet rate could offer any substantial benefit regarding a decision of trust. Due to this project's timeframe and scope of investigation, packet rate will be dropped and focus will be solely on power.

Each five second window, an array of power values will exist for each AP/STA but realistically, these values alone are not enough to represent a metric themselves. Consideration must then be given to how to use these values effectively to create a viable metric.

In order to create useable metrics, the values will be used to calculate an average power and variance value for the received power values every five seconds. Should an attack take place and an AP be spoofed, these metrics should offer some kind of representation that it is taking place. For example, variance is a measure of statistical dispersion between the values taken, thus, if an attack was taking place and the user was transmitting at a different power value to the legitimate AP then variance could offer some interesting insights.

In order to calculate these metrics, it's important to consider how the values are stored. The power values are stored in large arrays in their specific list node alongside the AP/STA's index value which indicates how many packets were captured for that node and in turn the number of values in the array.

The program will need to instigate a secondary thread, alongside the process packet thread, that will carry out the metric calculation. Essentially, this thread will carry out the following function.

- Sleep for five seconds
- Wake up and calculate the metrics for each node in each list
- Erase all the data from the last five seconds and reset index
- Sleep for another five seconds (and so on...)

The following code sample shows the secondary thread being instigated and then the main function for calculating the metrics.

```
main()
{
      ...rest of main function...

      pthread_create(&thread1, NULL, start_loop, NULL);
      pthread_create(&thread3, NULL, calculateMetrics, NULL);

}
```

```
void *calculateMetrics()
{

while(1)
{
	sleep(5);

	struct accessPoint *ap;
	pthread_mutex_lock( &G.mutex1 );
	list_for_each(&G.apList.accessPoints, ap, apList)
	{
		ap->avgpower = calculateMean(ap->pwr, ap->index);

		ap->varpower = calculateVariance(ap->pwr, ap->index);

		if(ap->varpower != 0)
		{
			ap->sdpower = sqrt((double)ap->varpower);
		}
		else ap->sdpower = 0;

		memset(ap->pwr, 0, sizeof(ap->pwr));

		ap->index = 0;

	}

	/*....insert same again for station list...*/

}
```

This thread is essentially an endless while loop that sleeps for five seconds upon each instigation. It then locks the thread mutex, as it will be reading from the data structures, and it instigates the *list_for_each( )* function to cycle through the list.

For each node, it uses the functions *calculateMean( )* and *calculateVariance( )* to calculate the two main metrics and it passes to these the array of power values and the index as arguments. The index is passed so that the function knows how many values are in the array. The variance is then checked to see if it is non-zero and if so, the Standard Deviation is also calculated as an additional value.

The *memset( )* function is then used to reset the array to zero while the index is also set to zero to begin collecting data for the next five second window.

The methods mentioned for calculating the mean and variance can be found in Appendix 1 and are not included here due to sheer simplicity.

8.3.5 Outputting to File and Printing a UI

Now that the metrics are being properly calculated for every five second window, there needs to be a way to store each of these window calculations permanently for analysis as well as presenting it to the user.

Storing the results is a relatively simple process as this is just done by writing out to a text file upon calculating the metrics. The index is also used to print the number of packets received for each AP/STA during that specific five second window and this will be used as a level of confidence when detecting an attack. When the software tool has been stopped, a text file will exist contained a readout of the values for each AP at every five second window. This is done as follows.

```
void *calculateMetrics()
{
while(1)
{
      sleep(5);

      file = fopen("results.txt","a+");
      fprintf(file, "%s", "ACCESS POINTS\n\n");
      list_for_each(&G.apList.accessPoints, ap, apList)
      {

            ...calculate metrics for AP list...

            fprintf(file, "%02x:%02x\n", ap->mac[4],ap->mac[5]);

            fprintf(file, "Pwr:\tAverage: %d\tVariance: %d\tStanDev:
            %lf #Pckts: %d\n\n",ap->avgpower,ap->varpower,ap->sdpower,
            ap->index);

      }

            fprintf(file, "%s", "USERS\n\n");

            ...calculate station metrics and use fprintf as above..
```

Printing a user interface (UI), on the other hand, is a slightly more difficult task but ultimately quite similar. What makes printing a UI slightly more difficult is that in order to do so, a third thread is going to have to be instigated to deal with this.

The code sample below displays this functionality.

```
main()
{
      ...rest of main function...

      pthread_create(&thread1, NULL, start_loop, NULL);
      pthread_create(&thread2, NULL, printGUI, NULL);
      pthread_create(&thread3, NULL, calculateMetrics, NULL);

}
```

```
void *printGUI()
{
	while(1)
	{

		system("clear");

		printf("ACCESS POINTS\n");
		printf("=============\n\n");

		struct accessPoint *ap;
		list_for_each(&G.apList.accessPoints, ap, apList)
		{

			printf("%02x:%02x\n", ap->mac[4],ap->mac[5]);


			printf("Pwr:\t\tAverage: %d\tVariance: %d \t
			StanDev: %lf\n",ap->avgpower, ap->varpower, ap-
			>sdpower);

		}

		printf("USERS\n");
		printf("=====\n\n");

		...print information about STAs as above...

		sleep(2);
	}
}
```

This basically just prints a title then uses the *list_for_each( )* function to cycle through a list and print each nodes MAC address, average power and power variance to the terminal screen for the user to see. Note that like the file print, this only prints the last two elements of the MAC address. Again this is due to the strange bug that was mentioned before.

As can be seen, this is contained within an endless while loop with a sleep function at the bottom. The *system("clear")* function at the beginning of the loop clears the terminal screen so that the next print looks like it updates the values on screen.

This provides an output quite like the following.

```
ACCESS POINTS
=============
4e:6f
Average: 96   Variance: 25    SD: 5
7f:21
Average: 81   Variance: 4     SD: 2

USERS
=====
65:fe
Average: 56   Variance: 64    SD: 8
```

8.4 Program Flow

Now that the software tool's design has been fully specified, the following diagram gives a full overview of the entire software tool's flow and functionality.

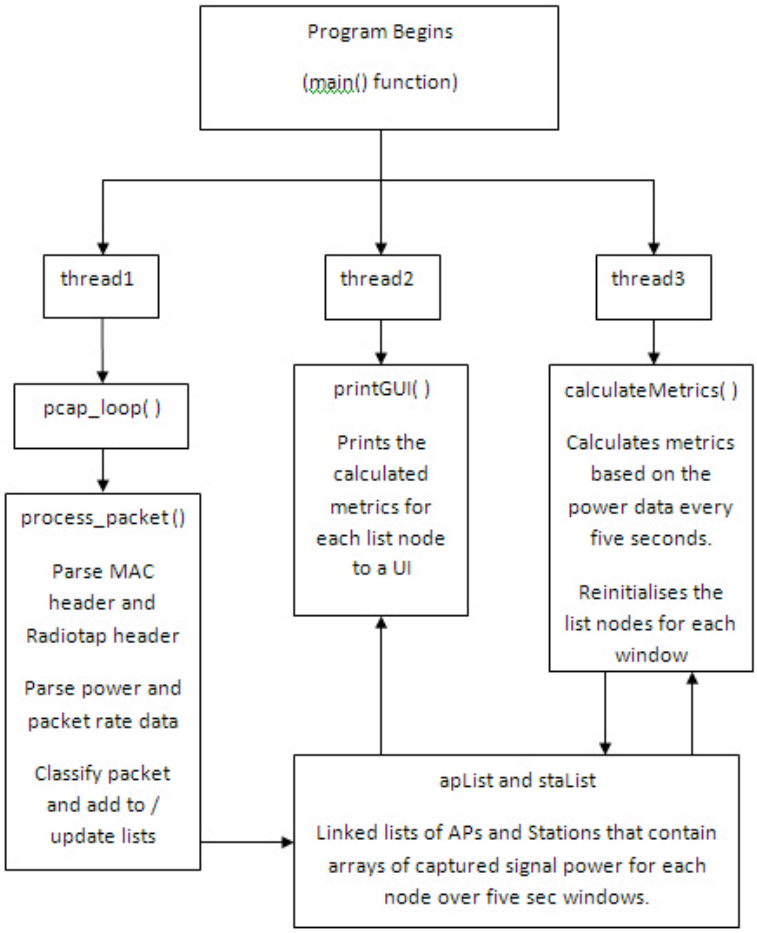


Figure 19 - Full Software Tool Diagram

9 Testing Packet Capture Tool for Determining Trust

Having developed a packet capture tool in software, it is important to spend some time testing to determine whether the recorded metrics could be utilised for determining the trustworthiness of a network.

At this point it is important to note that the tests carried out will be done solely on Access Points and the information regarding the stations will be disregarded within the scope of this project. The reasoning behind this is that if the threat model is considered once more, most, if not all, of the attacks involve the attacker "spoofing" an AP in order to carry out the attack. For example, in the Denial of Service (DoS) attack, the attacker assumes the identity of the AP and broadcasts a large number of deauthentication frames to the stations connected to that AP.

For this reason, the DoS attack will used for testing purposes as it is an extremely effective attack. Hopefully the metrics recorded over time will enable the attack to be recognised and a certain level of confidence offered that in fact, the AP has become untrustworthy.

These tests will be carried out under some different network circumstances and results presented over a period of time, each period containing a DoS attack.

9.1 Testing Plan

In order to test a small open network such as one that might be found in a small coffee shop or lounge, it is important to try and recreate those surroundings as best possible. The following is a plan view of the laboratory as it was set up to carry out these experiments.

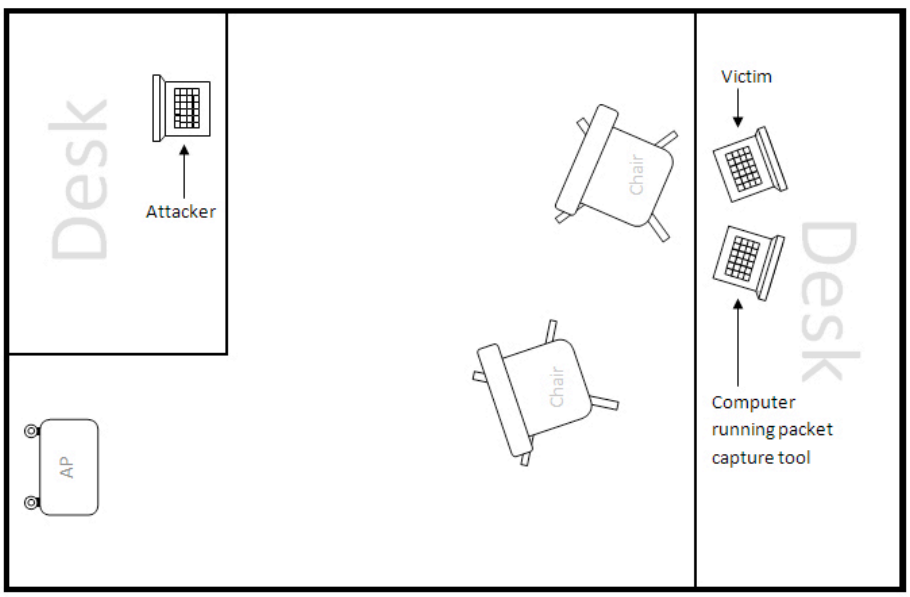


Figure 20 - Plan view of Lab for experiments

As can be seen, the attacker and the AP are at different locations and the victim of the attack is positioned at the other side of the room to both the AP and the attacker. The monitoring computer running the packet capture software will sit directly next to the victim computer. Ideally, it would be beneficial to run the monitoring software on the victim machine but due to some hardware issues when running multiple VAP's, it was decided to run the monitor next to the victim to receive better results. Essentially, as the monitor is in effectively the same place as the victim, the monitoring will be accurate.

This will hopefully not only show that the monitor can detect untrustworthiness, but that it can detect it when it is running on a machine other than the victim machine.

Two main tests will be carried out and these are as follows.

- DoS attack on victim operating with low levels of traffic
- DoS attack on victim operating with high levels of traffic

This will test the monitoring software under two very common network situations and will hopefully allow a strong indicator of trust to be viewed.

For each attack, the values of Average Power, Power Variance and the Number of Packets will be monitored over time for the AP in question and these will be graphed against time

9.2 DoS Attack on a Low Usage Station

The following graph displays the recorded metrics for a DoS attack on a victim that is displaying low usage. In order to create low usage, the victim was simply "pinging" the AP continuously.

The monitoring period is roughly 80 seconds and the attack itself takes place at roughly 45-50 seconds for a duration of 12-15 seconds.

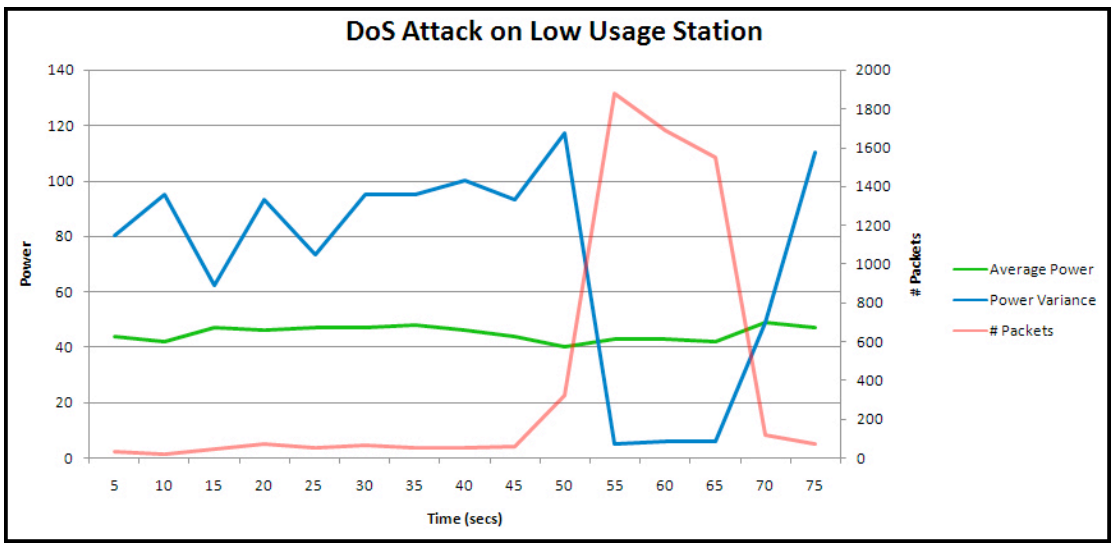


Figure 21 - Graph showing DoS attack on low usage victim

Initially, the graph shows an extremely low number of packets with a relatively constant average power and this continues for the initial 45 seconds of the monitoring period. Over this initial 45 second period, the variance fluctuates quite actively but consistently remains around the same figure.

At 45 seconds, roughly when the DoS attack takes place, the variance initially spikes almost as high as 120 while the number of packets from the AP begins to rise sharply. As the number of packets peaks at over 1800 around 55 seconds, the variance drops to below 10 and these both remain roughly in this region for the duration of the attack.

When the attack ends, variance rises back to roughly the same position as before while the number of packets does almost exactly the same and falls back to its original low value.

It's also worth noting that during the entire monitoring period, the average power remains consistently the same.

9.3 DoS Attack on a High Usage Station

The following graph displays the recorded metrics for a DoS attack on a victim that is operating with a high level of usage. In order to create a high usage situation, the victim was streaming an online video.

The monitoring period is roughly 60 seconds and the attack itself takes place at roughly 30-35 seconds for a duration of roughly 15 seconds.

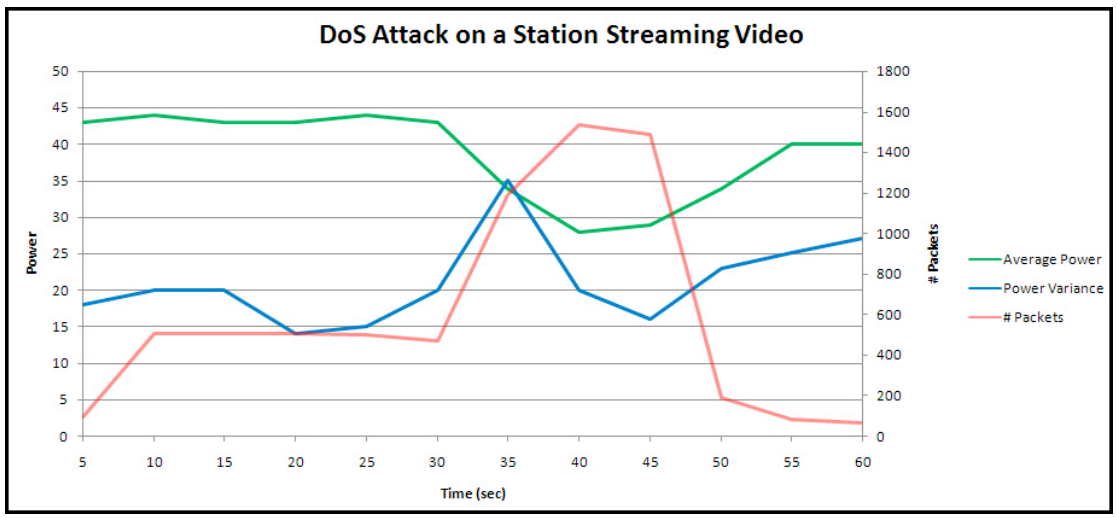


Figure 22 - Graph showing DoS attack on high usage victim

At the beginning of the monitoring period, the graph shows a slight rise in number of packets with a relatively constant average power of just below 45. This continues for 30 seconds of the monitoring period with the number of packets remaining at roughly 550. Over this initial period of 30 seconds, the variance remains relatively constant around 17.

At 30 seconds, roughly when the DoS attack occurs, the variance begins to spike, peaking at 35 roughly 5 seconds later. The number of packets from the AP begins to rise sharply at the same time, peaking at 40 seconds at nearly 1600 packets. The variance begins to taper off again while the attack is taking place and begins to rise slightly at the end of the attack. Across the entire duration of the attack window, the average power falls considerably to under 30.

When the attack ends, the number of packets falls back to a low value while the average power rises to just below its original value before the attack.

9.4 Results Discussion

Having observed these two scenarios, it is clear to see that during the window of time that the attack is taking place, the recorded metrics do change and display certain characteristics. It is important to consider these results and why they take this shape.

9.4.1 Scenario One: Low Usage Station

In this scenario, the attack took place at roughly 45 seconds and the strongest observable change is the massive downward spike in power variance.

Initially, the variance is quite high and fluctuates around 80. This can be attributed to the fact that the number of packets in the air from that AP is quite low. The station is simply pinging the AP and thus generating extremely small amounts of traffic and in this case, each packet is separated by a larger time. This may be why the variance fluctuates so much as with a lower number of packets they are being spread out more over time.

When the attack takes place, the variance spikes extremely high to 120 and then within a single monitoring window it drops to an extremely low value of around 6. This is quite easy to explain if the victim's usage and the attack are both considered.

The initial spike in variance that occurs in the 45-50 second monitoring window happens because the attack begins in this window and the attacker's frames are being mixed with the legitimate AP frames. Considering the idea that they both present different power values to the monitor, this would cause the variance of the captured packets to be high in this window. For the next three monitoring windows, the attacker is still broadcasting a large number of deauthentication frames but by this stage, the victim has already been de-authenticated from the AP so all of the captured frames are from the attacker. Considering that the attacker is sending a large number of these frames from the same location with very little space in between them would explain why the variance is low as there will be very little difference between the packets.

Also, as was observed, the number of packets from the AP is considerably low initially. This is simply because the victim is only pinging the AP in order to generate a low amount of traffic, thus the number of packets from the AP is low. At the 45 second monitoring window, the number of packets shoots up to over 1800 and remains this high until the 70 second monitoring window, which is in line with the DoS attack. This, again, is extremely easy to explain as when the attack takes place, the attacker is broadcasting a large number of deauthentication frames every single

second and thus, the number of packets appearing to come from the AP is going to be very high. At the time when the attack ends, the number of packets returns to its original low value as the attacker is no longer broadcasting a large number of frames and thus there are not as many packets appearing to be from the AP in the air.

The average power value remains around 40 for the duration of the entire monitoring period and does not show any change during the attack. This is difficult to explain as with such a shift in variance, a change in average power would have been expected. This could be attributed though to the fact that the victim is generating such a low amount of traffic that it is difficult to generate a viable figure for average power when an attack isn't taking place and it also may be that the attacking power just so happens to offer a similar average power.

9.4.2 Scenario Two: High Usage Station

Initially, the number of packets rises to roughly 500 and it is very clear that this is simply packets attributed to the usage of the victim who in this case is streaming an online video. Both power variance and average power remain relatively constant initially, at around 17 and 43 respectively. The variance does not fluctuate so much initially like before and this is because there are a lot more packets being sent from the AP to the victim with less time in between them. This means more packets are sent under the same radio conditions and the power values will have a more constant variance. Similarly, this explains the constant average power to an extent also.

When the attack takes place at around 30 seconds, three main observations occur. The variance initially spikes up over the processing two monitoring windows and then tapers off again over the duration of the attack, the average power drops off slowly and remains lower for the duration of the attack and the number of packets appearing from the AP spikes upwards sharply to roughly 1600.

The initial spike in the variance in the 30 second monitoring window can be attributed to the same reason as the previous scenario; the attack begins in this window and the attacker's frames are being mixed with the legitimate AP frames. This then tapers off to a more constant value after the beginning of the attack as the victim has become de-authenticated and the only packets being received from the AP are the attacker's spoofed packets.

The massive jump in the number of packets from 500 to 1600 in the 30 second monitoring window is at the beginning of the attack and so can be easily be explained as in the previous scenario. During the DoS attack the attacker generates a lot of deauthentication frames every second which will push this figure up. This figure remains this high for the duration of the attack and then falls off to a low value. This backs up the explanation and the reason that it doesn't return to 500 like it was before the attack is because the user has become de-authenticated and has not reconnected to stream the video.

In this scenario, the value for the average power drops from 45 to 30 and remains at this level for the duration of the attack. This is a more likely observation for average power than in the previous scenario as when an attacker broadcasts many frames appearing as the AP, it would be expected that these frames would be of a different power level, thus affecting the value of average power.

9.5 Viability of Metrics for Determining Trust

Given the observations that have been made regarding the monitored attacks, it is necessary to discuss whether the measured metrics could be used to determine trust.

In both scenarios, the power variance provides some interesting observations. In each case, as the attack begins, the variance displays a relatively large upward spike which is because the legitimate and spoofed frames are mixing together with different power values. The variance then evens out again but at a different value that previously. These are extremely observable characteristics that are common over both scenarios.

Also, while the average power doesn't offer much of an observation in the initial scenario, the second scenario shows a relatively large drop in average power for the duration of the attack. As was explained, this is because the received power from the attacker is more likely to be different than for the legitimate AP. This again presents a clearly observable metric.

While both of the mentioned metrics appear to be quite viable, this viability is accentuated if the number of packets from the access point is considered. This value can almost be considered as a confidence value for the decision of trust. For example, if observations such as those above were made with the average power and power variance then this could signal that an AP has been compromised. This being said, this decision could be made more or less confident based on the number of packets from that AP. If the above observations were made but did not occur alongside a spike in number of packets, then it's probably more likely that it is an anomalous glitch based on the medium. On the other hand if the observations were made and there was a spike in the number of packets then a more confident decision could be made that, in fact, an AP has become untrustworthy.

So as can be seen, the metrics do offer a considerable amount of viability for determining trust but for them to be realistically used, a certain level of classification would need to be carried out and maybe additional metrics considered alongside the ones observed.

10 Conclusions

10.1 Summary

This document has served to demonstrate the steps taken to investigate and implement a means of determining trust on a ubiquitous 802.11 network. In order to finalise appropriately, it is important to summaries the work carried out so that certain conclusions can be made and the success of the objectives discussed.

Initially, following a short investigation into P2P trust and reputation systems and the technical aspects of 802.11, it was decided that the threat model of a ubiquitous Wi-Fi network should be fully investigated by carrying out a range of 802.11. It was decided that following this, a Packet capture program would be written in C to utilise additional channel specific data to attempt to make a decision of trust for an AP.

Denial of Service (DoS), the cracking of WEP and WPA and man in the middle were fully investigated and for each attack, the methodology employed was detailed fully with a clear explanation of attacker behaviour in each case. This allowed a comprehensive threat model tobe fully developed.

This threat model was then built on to develop a packet capture tool in C using the libpcap programming library. Having examined the threat model it was proposed that the tool should specifically capture certain fields of the additional radiotap header data in order to examine certain channel specific characteristics and calculate metrics based on these. The tool's development process was outlined in detail and it was shown how this tool could be developed to capture specific packet data, namely the received power level of the packet.

It was then shown how this data was used to calculate a number of metrics that would divulge more pertinent information for making a trust decision. The project showed how the power values could be used to calculate average power and the variance of the power received based on data captured in every 5 second window. This detailed how it was important to calculate these metrics in every window based only on data received in that window in order to be accurate.

The investigation then put the packet capture tool to use by carrying out a number of experiments on a physical network. This displayed how each experiment presented some interestingly observable characteristics during the duration of an attack and it was shown that with some classification, these characteristics were extremely viable and could be used to determine a decision of trust.

Referring back to the Gantt chart in appendix 3 that was created as the project objectives were refined, it can be noted that this was mostly stuck to throughout the project and time management was not really as issue. The coding of the software tool did take a little longer than necessary though and this did unfortunately eat into the experimentation time slightly although now enough to create any kind of significant problem.

10.2 Review of Objectives and Main Conclusions

An initial objective of the project was to perform an investigation into the implementation of trust and reputation in P2P systems and as can be seen from the initial stages of this project, this has been successfully carried out. The early thought process of the project was drawn from the core thinking of P2P trust and Hector Molina's model of information gathering, ranking and response has been followed across the entirety of the project. This investigation was also extended to include 802.11 Ad Hoc deployments and key points regarding information gathering were discovered.

Following this, the objectives continue with a proposed investigation into the way illegitimate users can appear on a network. This objective was completely fulfilled in the implementation of the 802.11 attacks as with the examination and physical development of these attacks, the attacker became the scope of the investigation and thus their position and operation was deeply understood.

Given this study of the attacks, this allowed the next objective to be met which was to develop an algorithm to generate trust metrics. Having carried out the attacks, it was noted that the attacker performed a "spoof" of a legitimate AP in almost every one of the detailed attacks. This led to the thinking that if the received power from the AP could be recorded at the victim, then when the attacker masquerades themselves as the AP, metrics based on the received power will generate observable characteristics that would indicate an attack.

A packet capture tool was developed that parsed a power value from an appendage of each packet that contained additional channel specific details called radiotap data. This value was used to calculate an average power value and variance of power values within every 5 second window for each individual AP. This essentially allowed a number of accurate trust metrics to be generated for every five second window on the network.

Using this tool, the final two main objectives were completed by deploying the tool on a physical 802.11 network. This network was set up to be as close to a common, everyday deployment as possible and an attacker ran DoS attacks in two different scenarios. The DoS attack was chosen as it is an extremely common and easy to deploy attack that involves AP spoofing. In each case it was clear to see that the recorded metrics displayed highly observable characteristics when the attack was taking place. Section 9.4 fully outlines the reasoning behind these observations noting exactly why they are taking place.

Following on from 9.4, some discussion is offered as to how these recorded metrics could be used to determine trust. It is noted that they most definitely display likely characteristics when an attack takes place that are highly observable and thus would indicate the attack and the APs apparent untrustworthiness. It is also noted in 9.5 that the confidence of these observations is greatly increased when the number of packets received is high, thus providing another factor in the decision of trust.

An important point to note though, is that while the viability of using these metrics for determining the trustworthiness of an AP is relatively high, in order for it to be appropriately made, some effort would need to be placed in the development of a classification engine. This is further discussed in section 10.3.

To this point, the investigation has fulfilled all of the main objectives set forth although looking at some of the additional objectives, outlined in section 2.1, it can be noted that this project fulfils a number of these also. Section 9 as a whole witnesses the packet capture tool being experimented with on a realistic open network deployment that is set up to be as identical as possible to a network in somewhere like a cafe or a small lounge. This provides a realistic look at the metrics captured.

The graphs also record the AP metrics for lengthy periods of time in each case, recording metrics for each 5 second window of time as was discussed. This allows the metrics to be seen evolving over time, allowing them to be viewed in normal network situations and also while under attack. This displays that there are definitive differences between the two situations with regards to the metrics over time.

Having carried out these investigations, it is also important to mention that none of the methods for determining trust or detecting attacks that are employed in the field today actually use received power as a metric. As was mentioned in section 7.6, traffic patterns, statistical analysis and elaborate location sensing methods are used for detection of spoofing and attacks but it is very clear that none of the tools used utilise power.

This investigation, and its respective conclusions, has clearly displayed that by utilising metrics calculated from received power values, a very lightweight method of determining the trust of an AP, i.e. when it has been spoofed, can be achieved.

10.3 Future Work

Regardless of the relative success of this project, there are still a large number of areas that both require and suggest future work that can be carried out to continue this investigation.

Initially, deeper investigative experimentation should be carried out to fully ratify the findings of this small investigation. Due to both scope and time constraints, this project was simply able to carry out two small scenario based experiments but it order to further affirm to findings, it would be beneficial to load test the metrics with a lot of users accessing multiple APs in the same scenario.

Alongside these tests, it may be beneficial to begin to include additional metrics in the same way that this project has utilised power. There may be additional correlations that can be found in things like sequence number studies or packet rates or even in antenna information that when presented alongside power, could paint a clearer picture and help make a more affirmed decision.

Having recorded metrics and observed them in attack situations, it has been noted that a certain amount of classification would be required in order to carry out a decision of trust. Possibly, a classification engine could be employed to observe metrics in real time and make this decision. The engine could be “taught” what kind of observed metrics offer stronger and more confident signs that others and the system can learn when to deem activity untrustworthy. This is no small task though and represents an entirely separate layer of work that can only come after deep investigation into all metrics.

A possible additional piece of work would be to investigate and develop a sophisticated way of legitimate users on a network to monitor and share data in real time with either each other or a central system. This may tie in quite nicely with an entire classification engine deployment and offer a strong method of helping with the overall trust of a network.

11 References and Bibliography

[1]: Taxonomy of Trust by Hector Molina and Sergio Marti (Computer Networks Volume 3)

[2]: CORE: A Collaborative Reputation Mechanism to enforce node cooperation in Mobile Ad hoc Networks by Pietro Michiardi and Refik Molva

[3]: Addressing Wireless Threats in the Cisco Unified Wireless LAN: A Cisco whitepaper from http://www.cisco.com

[4]: Breezecom Technical Tutorial on 802.11 by Pablo Brenner

[5]: IEEE 802.11 handbook by Bob O'Hara and Al Petrick

[6]: Hacking Exposed Wireless by Johnny Cache, Vincent Liu

[7]: http://www.radiotap.org

[8]: http://madwifi-project.org/

[9]: http://www.aircrack-ng.org

[10]: Using the Radiotap Iterator: http://www.mjmwired.net/kernel/Documentation/networking/radiotap-headers.txt

[11]: http://ccan.ozlabs.org/info/list.html

Appendix 1

The following appendix contains a full code listing for the packet capture tool that was outlined in section 8.

The following files are included in this listing: *pktsniffer.c, packet.c, pktsniffer.h, 802_11.h* and the Makefile used for compilation.

The following files are not included as they have been integrated from open source locations and as such are referenced accordingly in the document: *list.c, list.h, ieee80211_radiotap.h, radiotap.c* and *radiotap.h.*

```
/*
pktsniffer.c

This C file contains entry point for program, thread
set up and functions to begin each thread.

*/

#include <stdio.h>
#include <stdint.h>
#include <pcap.h>
#include <pthread.h>
#include "802_11.h"
#include "pktsniffer.h"
#include <math.h>
#include "ccan/list/list.h"
#include <string.h>

/****** FUNCTION PROTOTYPES **********/

void *start_loop();
int calculateMean(int values[], int n);
int calculateVariance(int values[], int n);
void *printGUI();
void process_packet(u_char *whatever, const struct pcap_pkthdr *pkthdr,
const u_char* packet);
void *calculateMetrics();

/* Additional Variables */

struct pcap_pkthdr  header;
const u_char        *packet;
int                 i;
FILE                *file;

globals             G;


main()
{
       char errbuf[PCAP_ERRBUF_SIZE];

       /* Creating device handle and opening device */
       G.handle = pcap_open_live("ath1", 2048, 1, 1000, errbuf);
       if (G.handle == NULL)
       {
```

```
                fprintf(stderr, "Couldn't open device: %s\n", errbuf);
                return(2);
        }

        /* List creation and initialisation */
        list_head_init(&G.apList.accessPoints);
        list_head_init(&G.staList.stations);


        pthread_t thread1, thread2, thread3;

        /* Thread mutex used for locking data structures */
        pthread_mutex_init(&G.mutex1, NULL);

        /* Creation of three threads */
        pthread_create(&thread1, NULL, start_loop, NULL);
        pthread_create(&thread2, NULL, printGUI, NULL);
        pthread_create(&thread3, NULL, calculateMetrics, NULL);


        pthread_join( thread1, NULL);
        pthread_join( thread2, NULL);


}

/*
*start_loop()

Fucntion for packet capture thread - instigates the
pcap_loop.
*/
void *start_loop()
{
        pcap_loop(G.handle, -1, process_packet, NULL);
        pcap_close(G.handle);
}

/*
calculateMean()

fucntion to calculate the mean of 'n' values of an array
*/
int calculateMean(int values[], int n)
{
        if (n==0)
        {
                return 0;
        }

        int sum = 0;
        int i;
        for (i = 0; i < n; i++)
        {
                sum += values[i];
        }
        return sum / n;
}

/*
calculateVariance()

fucntion to calculate the variance of 'n' values of an array
*/
int calculateVariance(int values[], int n)
{
```

```
	if (n==0 | n==1)
	{
		return 0;
	}

	int valuesMean = calculateMean(values, n);
	int sum = 0;
	int i;
	for (i = 0; i < n; i++)
	{
		sum += (values[i] - valuesMean) * (values[i] - valuesMean);
	}
	return sum / (n-1);
}

/*
*printGUI()

Fucntion for UI thread - prints results to user/
*/
void *printGUI()
{
	while(1)
	{

		system("clear");

		printf("ACCESS POINTS\n");
		printf("=============\n\n");

		struct accessPoint *ap;
		/* Look through each element of the AP list and print details
		for each */
		list_for_each(&G.apList.accessPoints, ap, apList)
		{
			printf("%02x:%02x\n", ap->mac[4],ap->mac[5]);

			printf("Pwr:\t\tAverage: %d\tVariance: %d\tStanDev:
			%lf\n",ap->avgpower, ap->varpower, ap->sdpower);

		}
	 printf("USERS\n");
	 printf("=====\n\n");

	 struct station *sta;
		/* Look through each element of the STA list and print details
		for each */
		list_for_each(&G.staList.stations, sta, staList)
		{
			printf("%02x:%02x\n", sta->mac[4],sta->mac[5]);

			printf("Pwr:\t\tAverage: %d\tVariance: %d\tStanDev:
			%lf\n",sta->avgpower,sta->varpower, sta->sdpower);
		}

		sleep(2);

	}
}

/*
*calculateMetrics()

Fucntion for Calculation thread  - calculates metrics, stores them
and resets monitoring window storage
*/
```

```
void *calculateMetrics()
{
	while(1)
	{
		sleep(5);

		file = fopen("results.txt","a+");
		fprintf(file, "%s", "ACCESS POINTS\n\n");
		struct accessPoint *ap;
		pthread_mutex_lock( &G.mutex1 );

		/* Look through each element of the AP list and calculate
		metrics for each */
		list_for_each(&G.apList.accessPoints, ap, apList)
		{
			ap->avgpower = calculateMean(ap->pwr, ap->index);
			ap->avgpower_w = calculateMean(ap->pwr_w, ap->index);

			ap->varpower = calculateVariance(ap->pwr, ap->index);
			ap->varpower_w = calculateVariance(ap->pwr_w, ap-
			>index);

			if(ap->varpower != 0)
			{
				ap->sdpower = sqrt((double)ap->varpower);
			}
			else ap->sdpower = 0;

			if(ap->varpower_w != 0)
			{
				ap->sdpower_w = sqrt((double)ap->varpower_w);
			}
			else ap->sdpower_w = 0;

			fprintf(file, "%02x:%02x\n", ap->mac[4],ap->mac[5]);
			fprintf(file, "Pwr:\tAverage: %d\tVariance: %d\tStanDev:
			%lf\n\n",ap->avgpower,ap->varpower,ap->sdpower);


			memset(ap->pwr, 0, sizeof(ap->pwr));
			memset(ap->pwr_w, 0, sizeof(ap->pwr_w));
			ap->index = 0;

		}

		fprintf(file, "%s", "USERS\n\n");

		struct station *sta;

		/* Look through each element of the STA list and calculate
		metrics for each */
		list_for_each(&G.staList.stations, sta, staList)
		{
			sta->avgpower = calculateMean(sta->pwr, sta->index);
			sta->avgpower_w = calculateMean(sta->pwr_w, sta->index);

			sta->varpower = calculateVariance(sta->pwr, sta->index);
			sta->varpower_w = calculateVariance(sta->pwr_w, sta-
			>index);

			if(sta->varpower != 0)
			{
				sta->sdpower = sqrt((double)sta->varpower);
			}
			else sta->sdpower = 0;
```

```
                        if(sta->varpower_w != 0)
                        {
                                sta->sdpower_w = sqrt((double)sta->varpower_w);
                        }
                        else sta->sdpower_w = 0;

                        fprintf(file, "%02x:%02x\n", sta->mac[4],sta->mac[5]);
                        fprintf(file, "Pwr:\tAverage: %d\tVariance: %d\tStanDev:
                        %lf\n\n",sta->avgpower,sta->varpower, sta->sdpower);

                        memset(sta->pwr, 0, sizeof(sta->pwr));
                        memset(sta->pwr_w, 0, sizeof(sta->pwr_w));
                        sta->index = 0;

                }
                pthread_mutex_unlock( &G.mutex1 );
                fprintf(file, "%s", "\n\n*****************************\n\n");
                fclose(file);
        }
}
```

```
/*
packet.c

This C file contains functions to process the packet,
classify it and store the captured data to lists

*/

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <pcap.h>
#include "ieee80211_radiotap.h"
#include <errno.h>
#include "radiotap.h"
#include "802_11.h"
#include "ccan/list/list.h"
#include "pktsniffer.h"
#include <pthread.h>
#include <string.h>
#include <math.h>

int count=0;

/****** FUNCTION PROTOTYPES **********/

void process_packet(u_char *whatever, const struct pcap_pkthdr *pkthdr,
const u_char* packet);
int isFrame_ap(const struct header *frame);
int inAPlist(u_char address[6]);
int inSTAlist(u_char address[6]);
void process_rtap(struct ieee80211_radiotap_header* rtap_header);

extern globals G;

/*
process_packet()

Callback function for pcap_loop
*/
```

```
void process_packet(u_char *whatever, const struct pcap_pkthdr *pkthdr,
const u_char* packet)
{
	/* Parse frame */
	struct ieee80211_radiotap_header *radiotap;
	radiotap = (struct ieee80211_radiotap_header*) packet;

	const struct header *frame;

	frame = (struct header*) (packet+radiotap->it_len);

	process_rtap(radiotap);

	/* Classify frame */
	if(isFrame_ap(frame))
	{
		/* The frame is an AP but not in the list */
		if(!(inAPlist(frame->address2)))
		{
			pthread_mutex_lock( &G.mutex1 );
			G.accessPoint = malloc(sizeof(*G.accessPoint));

			/* Copy data to new AP struct */
			memcpy(G.accessPoint->mac, frame->address2, 6);
			G.accessPoint->index = 0;
			G.accessPoint->pwr[G.accessPoint->index] = G.pwr;

			/* Add this to list */
			list_add_tail(&G.apList.accessPoints,
		&G.accessPoint->apList);
			G.apList.num_accessPoints++;

			pthread_mutex_unlock( &G.mutex1 );
		}
		/* The frame is an AP and in the list */
		else
		{
			pthread_mutex_lock( &G.mutex1 );
			/* Search through list of APs */
			list_for_each(&G.apList.accessPoints, G.accessPoint,
			apList)
			{
				/* when the AP is found update its data and
				increse index */
				if(G.accessPoint->mac[4] == frame->address2[4] &&
				G.accessPoint->mac[5] == frame->address2[5])
				{
					G.accessPoint->index++;
					G.accessPoint->pwr[G.accessPoint->index] =
					G.pwr;

					if(G.accessPoint->index == NO_PACKETS)
					{
						G.accessPoint->index = 0;
					}
				}
			}
			pthread_mutex_unlock( &G.mutex1 );

		}
	}
	else if(!(isFrame_ap(frame)))
	{
		/* The frame is an STA but not in the list */
		if(!(inSTAlist(frame->address2)))
```

```
		{
			pthread_mutex_lock( &G.mutex1 );
		G.station = malloc(sizeof(*G.station));

			/* Copy data to new STA struct */
			memcpy(G.station->mac, frame->address2, 6);
			G.station->index = 0;
			G.station->pwr[G.station->index] = G.pwr;

			/* Add this to list */
			list_add_tail(&G.staList.stations, &G.station->staList);
			G.staList.num_stations++;

			pthread_mutex_unlock( &G.mutex1 );

	}
		/* The frame is an STA and in the list */
		else
		{
			pthread_mutex_lock( &G.mutex1 );
			/* Search through list of APs */
			list_for_each(&G.staList.stations, G.station, staList)
			{
				/* when the AP is found update its data and
				increse index */
				if(G.station->mac[4] == frame->address2[4] &&
				G.station->mac[5] == frame->address2[5])
				{
					G.station->index++;
					G.station->pwr[G.station->index] = G.pwr;

					if(G.station->index == NO_PACKETS)
					{
						G.station->index = 0;
					}
				}
			}
			pthread_mutex_unlock( &G.mutex1 );
		}
	}
}


/*
isFrame_ap()

Function to determine whether a frame is from an AP or not
*/
int isFrame_ap(const struct header *frame)
{
	if(frame->frame_ctl[1] & (0x40) == 0x40)
	{
		return 0;
	}
	else
	{
		return 1;
	}

}

/*
inAPlist()

Function to determine whether an AP is already in the list
*/
```

```
int inAPlist(u_char address[6])
{
	struct accessPoint *ap;
	pthread_mutex_lock( &G.mutex1 );
	list_for_each(&G.apList.accessPoints, ap, apList)
	{
		if(ap->mac[4] == address[4] && ap->mac[5] == address[5])
		{
			pthread_mutex_unlock( &G.mutex1 );
		return 1;
		}
	}

	pthread_mutex_unlock( &G.mutex1 );
	return 0;
}

/*
inSTAlist()

Function to determine whether an STA is already in the list
*/
int inSTAlist(u_char address[6])
{
      struct station *sta;
	pthread_mutex_lock( &G.mutex1 );
	list_for_each(&G.staList.stations, sta, staList)
	{
		if(sta->mac[4] == address[4] && sta->mac[5] == address[5])
		{
			pthread_mutex_unlock( &G.mutex1 );
			return 1;
		}
	}
	pthread_mutex_unlock( &G.mutex1 );
	return 0;
}

/*
process_rtap()

Function to process radiotap header and
extract any requested data
*/
void process_rtap(struct ieee80211_radiotap_header* rtap_header)
{
	/* Initialise globals */
	G.pkt_rate_100kHz = 0, G.antenna = 0, G.pwr = 0;
	struct ieee80211_radiotap_iterator iterator;

	/* Initialise iterator */
	int ret = ieee80211_radiotap_iterator_init(&iterator, rtap_header,
rtap_header->it_len);

	while (!ret)
	{

		ret = ieee80211_radiotap_iterator_next(&iterator);

		if (ret)
			continue;

		/* Use switch to see if this argument is something we can use
		*/

		switch (iterator.this_arg_index) {
```

```
                case IEEE80211_RADIOTAP_RATE:
                                G.pkt_rate_100kHz = (*iterator.this_arg) * 5;
                                break;

                case IEEE80211_RADIOTAP_ANTENNA:
                                G.antenna = (*iterator.this_arg);
                                break;

                case IEEE80211_RADIOTAP_DB_ANTSIGNAL:
                                G.pwr = (*iterator.this_arg);
                                break;

                default:
                                break;
                }
        }
}
```

```
/*
pktsniffer.h

This C HGeader file contains the structures for APs and STAs
and also the list structs and a set of global variables

*/

#include "ccan/list/list.h"
#include <pthread.h>

#define NO_PACKETS          70000

struct apList{

struct list_head    accessPoints;
unsigned int        num_accessPoints;

};

struct accessPoint{

u_char              mac[6];
int                 pwr[NO_PACKETS];
int                 index;
int                 pwr_w[NO_PACKETS];
int                 avgpower, varpower;
double              sdpower;
int                 avgpower_w, varpower_w;
double              sdpower_w;
struct list_node    apList;

};

struct staList{

struct list_head    stations;
unsigned int        num_stations;

};

struct station{

u_char              mac[6];
int                 pwr[NO_PACKETS];
int                 index;
```

```
int                         pwr_w[NO_PACKETS];
int                         avgpower, varpower;
double                      sdpower;
int                         avgpower_w, varpower_w;
double                      sdpower_w;
struct list_node            staList;

};


typedef struct globals{

pcap_t                              *handle;
int                                 pkt_rate_100kHz, pwr, antenna, count,
                                    varpower, varpktrate, avgpower, avgpktrate;

double                              sdpower, sdpktrate;

struct apList                       apList;
struct accessPoint                  *accessPoint;

struct staList                      staList;
struct station                      *station;

pthread_mutex_t                     mutex1;

}globals;
```

```
/*
802_11.h

This C header file contains a representation
of the MAC header of an 802.11 frame.

*/

struct header{
       u_char       frame_ctl[2];
       u_char       duration_id[2];
       u_char       address1[6];
       u_char       address2[6];
       u_char       address3[6];
       u_char       seq_ctl[2];
       u_char       address4[6];
};
```

Appendix 2

The following tables contain the data gathered during the experiements in section 9 and were used to compile the graphs that are seen in figures 21 and 22.

DoS attack on station watching video

Window (secs)	Average Power	Power Variance	SD	# Packets
5	43	18	4.242	91
10	44	20	4.472	502
15	43	20	4.47	504
20	43	14	3.74	502
25	44	15	3.87	496
30	43	20	4.47	467
35	34	35	5.91	1188
40	28	20	4.47	1536
45	29	16	4	1486
50	34	23	4.79	186
55	40	25	5	79
60	40	27	5.19	63

DoS attack on pinging station from same station

Window (secs)	Average Power	Power Variance	SD	# Packets
5	44	80	8.94	34
10	42	95	9.74	18
15	47	62	7.87	47
20	46	93	9.64	72
25	47	73	8.54	54
30	47	95	9.74	65
35	48	95	9.746	54
40	46	100	10	50
45	44	93	9.64	59
50	40	117	10.81	321
55	43	5	2.23	1881
60	43	6	2.45	1689
65	42	6	2.449	1551
70	49	49	7	119
75	47	110	10.48	74

Appendix 3

Below is the Gantt chart created for the project in question.