LETTERKENNY INSTITUTE OF TECHNOLOGY

A thesis submitted in partial fulfilment of the
Requirements for the Master of Science in Computing in
Systems and Software Security Letterkenny Institute of Technology

Improving Compliance With Bluetooth Device Detection

Author: *Martin Davies*

Supervisor: *Dr. Eoghan Furey*

Submitted in May 2015 to

Quality and Qualifications Ireland (QQI)

Dearbhú Cáilíochta agus Cáilíochtaí Éireann

Declaration

I hereby certify that the material, which I now submit for assessment on the programs of study leading to the award of Master of Science in Computing in System and Software Security, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Signature of candidate: _____________________________ Date: _________________

Acknowledgements

This thesis would not have been possible without the support of many people at Letterkenny Institute of Technology. The author would like to acknowledge and express his sincere gratitude to his supervisor, Dr. Eoghan Furey, who was abundantly helpful and offered invaluable support and guidance.

The author would also like to thank the academic staff who contributed to the MSc course; John O'Raw, John McGarvey, Nigel McKelvey, William Farrelly and Mark Leeney. Recognition is due to my colleague Anthony Caldwell for offering guidance in the early stages, and to my classmates/colleagues Alan Martin Sweeney, Fergal Coll, Kelly McWeeney and Randolph Rosenberg.

Dominic Spill, who is cited several times in this thesis, should be acknowledged for quickly responding to some technical emails relating to the ubertooth-scan utility options.

Recognition is due to several U.S. colleagues; Andrea Cantatore, for sourcing vital Bluetooth equipment essential for completing this thesis, Bill Cardwell for help in sourcing Antenna equipment, and Glenn Hanlon, for sharing his own Bluetooth knowledge so freely.

The author wishes to thank his parents, Ken and Bridie Davies, for their support, love and encouragement. Thanks also to Stephen, Richard, Helen, Christine and their families for their support, and Eoin O'Brien for providing several research related books.

The author wishes to express his heartfelt love and gratitude to his wife Edel, son Dallan and daughter Fiadh, for their patience and understanding over the past three years of this MSc journey.

Finally, the author would like to dedicate this work to the memory of his Grandfather, Bryn Davies, who passed away during its writing.

Abstract

Attributed to Joshua Wright (2012) “Security will not get better until tools for practical exploration of the attack surface are made available”.

With Bluetooth enabled but discovery mode turned off, auditing for Bluetooth devices, or creating an accurate Bluetooth device hardware log has been limited. The software tools and hardware devices to monitor WiFi networking signals have long been a part of the security auditor’s arsenal, but similar tools for Bluetooth were bespoke, expensive, and beyond the scope of most security professionals.

However, this has changed with the introduction of the Ubertooth One, a low-cost and open-source platform for monitoring Bluetooth Classic signals. Using a combination of the Ubertooth One, and other high power Bluetooth devices, an auditor should now finally be able to actively scan for rogue devices that may otherwise have been missed.

This thesis looks at various hardware combinations that can be used to achieve this functionality, and the possible implications from a compliance point of view, with a particular focus on the standards used by the *Payment Card Industry Data Security Standard* (PCI-DSS), and the guidelines offered by the *National Institute of Standards and Technology* (NIST).

This work attempts to compare the results of scanning with traditional Bluetooth devices alone, compared to an Ubertooth/Bluetooth combination. Highlighting how this newfound ability to monitor a larger portion of Bluetooth traffic can potentially highlight serious implications in the compliance landscape of many organisations and companies.

The number of devices containing Bluetooth chipsets will continue to rise and this area of research will become more and more relevant as security and compliance auditors attempt to stem the tidal wave of vulnerabilities brought by the *Bring Your Own Device* (BYOD) and *Internet of Things* (IoT) phenomena.

Contents

List of Figures

List of Tables

List of Abbreviations

ACL	Asynchronous Connection-oriented Logical
AES	Advanced Encryption Standard
AFH	Adaptive Frequency Hopping
AP	Access Point
ARM	Advanced RISC Machine
BASH	Bourne Again Shell
BD_ADDR	Bluetooth Device Address
BLE	Bluetooth Low Energy
BNEP	Bluetooth Network Encapsulation Protocol
BR	Basic Rate
BYOD	Bring Your Own Device
CCM	Counter with CBC-MAC
CDE	Cardholder Data Environment
CRC	Cyclic Redundancy Check
CSR	Cambridge Silicon Radio (chipset)
CSV	Comma Separated Value/Variable
CVSD	Continuously Variable Slope Delta
DPSK	Differential Phase Shift Keying
DIDS	Distributed Intrusion Detection System
DOS	Denial of Service
ECDH	Elliptic Curve Diffie-Hellman (exchange)
EDR	Enhanced Data Rate
ERTM	Enhanced ReTransmission Mode
eSCO	Extended Synchronous Connection Orientated communications link
FEC	Forward Error Coding

FHSS	Frequency Hopping Spread Spectrum
GAO	Government Accountability Office
GAP	Generic Access Protocol
GFSK	Gaussian Frequency Shift Keying
HCI	Host Controller Interface
HID	Human Interface Device
HIDS	Host-based Intrusion Detection System
HS	High Speed
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IPS	Intrusion Prevention System
ISM	Industrial, Scientific and Medical
ISO	International Organisation for Standardisation
JTAG	Joint Test Action Group
L2CAP	Logical Link Control and Adaption Protocol
LAN	Local Area Network
LAP	Lower Address Part
LED	Light Emitting Diode
LMP	Link Manager Protocol
LTK	Long Term Key
LYIT	Letterkenny Institute of Technology
MAC	Media Access Control
MDM	Mobile Device Management
MITM	Man in the Middle
MTU	Maximum Transmission Unit
NAP	Network Access Point
NAP	Non-significant Address Part

NIDS	Network-based Intrusion Detection System
NIST	National Institute of Standards and Technology
OBEX	OBject EXchange
OUI	Organisationally Unique Identifier
PAN	Personal Area Network
PANU	Personal Area Network User
PCB	Printed Circuit Board
PCI-DSS	Payment Card Industry Data Security Standard
PDA	Personal Digital Assistant
PPM	Parts Per Million (a unit of wavelength accuracy value)
PPP	Point to Point Protocol
PSK	Phase Shift Keying
PSM	Protocol Service Multiplexor
RF	Radio Frequency
RFCOMM	Radio Frequency Communications Protocol
RISC	Reduced Instruction Set Computing
RP SMA	Reverse Polarity Sub-Miniature version A
RSSI	Received Signal Strength Indication
RX	Receiver
SCO	Synchronous Connection Orientated (communications link)
SDAP	Service Discovery Application Profile
SDP	Service Discovery Protocol
SDR	Software Defined Radio
SIEM	Security Information and Event Management
SIG	Special Interest Group
SM	Streaming Mode
SP	Special Publication
SSP	Secure Simple Pairing

TX	Transmitter
UAP	Upper Address Part
USB	Universal Serial Bus
UUID	Universal Unique IDentifier
USRP	Universal Software Radio Peripheral
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

CHAPTER 1 - Introduction

1.1 Purpose

This thesis is the final dissertation for the MSc in Systems and Software Security. This paper will demonstrate how Bluetooth technologies can now be actively monitored through the use of an Ubertooth One in conjunction with additional Bluetooth devices, and how this can impact the compliance landscape of an organisation.

1.2 Background

Now over twenty years old and well established, the volume of Bluetooth devices is constantly growing, and due to the nature of the technology have been difficult to monitor. Until recently, Bluetooth has relied upon security by obscurity. "Bluetooth technology is making its way into all kinds of devices, and is especially attractive due to its low cost and minimal resource requirements. Devices such as Bluetooth *Access Points* (AP) are available that provide similar connectivity and range as their 802.11 counterparts, but escape analysis mechanisms since Bluetooth operates using *Frequency Hopping Spread Spectrum* (FHSS) instead of traditional 802.11 transmission mechanisms" (Wright, 2007). With the introduction of the Ubertooth One, this monitoring and analysis can now take place in a cost-effective and efficient manner.

According to Wright (2011), "with the Ubertooth's ability to capture the lower 4 bytes of the *Bluetooth Device Address* (BD_ADDR), a standard Bluetooth dongle could be used to actively enumerate identified Bluetooth devices, the combination of the two *Universal Serial Bus* (USB) devices would provide the needed information to quickly and accurately characterise devices in the area". This device capture and categorisation is very important, "A rogue AP is any device that adds an unauthorised (and therefore unmanaged and unsecured) WLAN to the organisation's network" (Wireless Special Interest Group; PCI Security Standards Council, 2011). This thesis tests the above assertions, through the use of various USB Bluetooth devices in combination with the Ubertooth One ranging from the built-in low end device, through to a high power industrial device, exploring the important capability of creating a usable hardware inventory. As an ad-hoc technology, Bluetooth devices are often utilised within organisations, outside of the control of IT management. Few organisations recognise the threat of Bluetooth technology, often due to misconceptions in the technology, and the threats of use. Now armed with a hardware inventory appended with previously unavailable Bluetooth devices, it should be possible for the audit professional to identify known APs and Bluetooth stations from rogue devices. By doing this, this thesis aims to capture and identify non-discoverable devices, and reveal that a significant proportion of additional devices, with Bluetooth enabled, can now be captured, highlighting a gap in existing Bluetooth auditing practices.

1.3 Thesis Organisation

CHAPTER 2 - Literature Review. Appraises the background of the Bluetooth technology, and introduces compliance standards and guidelines.

CHAPTER 3 - Software and Hardware Requirements. Examines the software and hardware required for the experimentation stage of this thesis.

CHAPTER 4 - Design of Experiment. Outlines and describes how the experimentation was performed during each cycle.

CHAPTER 5 - Results and Discussion. Analyses the results of the testing and discusses the findings.

CHAPTER 6 - Conclusions and Further Research. Closes the thesis and suggests further research avenues.

CHAPTER 2 - Literature Review

2.1 Introduction

“Bluetooth technology has become all-pervasive, with attach rates close to one hundred percent for mobiles and laptops” (Gupta, 2013). Even though it is pervasive, there are misconceptions around range and exposure, so many organisations overlook potential threats. These threats should be taken seriously, and evaluated as part of an overall wireless security plan. This chapter introduces the Bluetooth specification, from its origins and specifications to its limitations, security issues and vulnerabilities. A brief description and outline of the purpose of intrusion detection systems is then covered, and finally compliance is introduced, and in particular the aspects of PCI-DSS and NIST guidelines that are affected through the use of Bluetooth technologies.

2.2 Bluetooth Overview

At the time of writing, Bluetooth is a stable, well documented and well established technology. Bluetooth dates back to 1994 when Ericsson came up with a idea to use a wireless connection to connect items such as an earphone and a cordless headset and the mobile phone. A couple of years later in 1998, five companies (Ericsson, Nokia, IBM, Toshiba and Intel) formed the Bluetooth *Special Interest Group* (SIG).

Bluetooth was originally intended to be a short range wireless technology, which primary purpose was to replace wires and cables. It has found uses with wireless keyboards, mice, headsets, hands-free kits, allowing mobiles communicate with computers, headphones, etc. The list of possible uses is endless. Bluetooth may not be the best choice for every wireless job out there, but it does excel at short-range cable-replacement-type applications. However, despite being considered a short range technology, the quantity of devices is enormous, and range distances can be quite significant. The number of devices is continually growing, with Statista (2015) estimating that the number of devices will increase to almost 10 billion by 2018.

2.2.1 Bluetooth Classic

The *Institute of Electrical and Electronics Engineers* (IEEE) created a family of standards called IEEE802 that deal with *Local Area Networks* (LAN). Two of the most relevant working groups of this standard are 802.11, which deals with wireless LANs, and 802.15 which specifies *Wireless Personal Area Networks* (WPAN). Bluetooth operates within the 802.15 standard, and uses the same frequency range (2.4 GHz) as 802.11 (WiFi).

Bluetooth uses the licence free *Industrial, Scientific and Medical* (ISM) frequency band for its radio signals and enables communications between devices up to a maximum distance of about 100m, although it is normally used for shorter distances (Poole, 2007). The Bluetooth channels are spaced 1MHz apart, beginning at 2402MHz and ending at 2480MHz. This arrangement of 79 individual Bluetooth channels gives a guard band of 2MHz at the bottom and 3.5MHz at the top which is presented in Table 1. It should be noted that the 2472MHz and 2480MHz bands are outside the standard operating frequencies for WiFi (in the US), and are highlighted in red.

Chl	MHz	Chl	MHz	Chl	MHz	Chl	MHz	Chl	MHz	Chl	MHz	Chl	MHz	Chl	MHz
1	2402	11	2412	21	2422	31	2432	41	2442	51	2452	61	2462	71	**2472**
2	2403	12	2413	22	2423	32	2433	42	2443	52	2453	62	2463	72	2473
3	2404	13	2414	23	2424	33	2434	43	2444	53	2454	63	2464	73	2474
4	2405	14	2415	24	2425	34	2435	44	2445	54	2455	64	2465	74	2475
5	2406	15	2416	25	2426	35	2436	45	2446	55	2456	65	2466	75	2476
6	2407	16	2417	26	2427	36	2437	46	2447	56	2457	66	2467	76	2477
7	2408	17	2418	27	2428	37	2438	47	2448	57	2458	67	2468	77	2478
8	2409	18	2419	28	2429	38	2439	48	2449	58	2459	68	2469	78	2479
9	2410	19	2420	29	2430	39	2440	49	2450	59	2460	69	2470	79	**2480**
10	2411	20	2421	30	2431	40	2441	50	2451	60	2461	70	2471		

Table 1: Bluetooth Channels and their respective MHz band

Bluetooth devices transmit in the 2.4GHz band using *Frequency-Hopping Spread Spectrum* (FHSS). Frequency hopping is a novel way to avoid busy channels used by other devices, WiFi or microwave ovens for example. A Bluetooth transmission remains only on a given frequency for a short time, unlike WiFi for example, and if any interference is present the data will be re-sent later when the signal has changed to a different channel which is likely to be clear of other interfering signals. The standard uses a hopping rate of 1600 hops per second, and the system hops over all the available frequencies using a pre-determined pseudo-random hop sequence based upon the Bluetooth address of the master node in the network.

Bluetooth hardware is rated in classes, which regulates the transmission output power, and therefore range of the devices. The common power classes are displayed in Table 2. Note that one device, the Aircable Host XR, described in more detail in section 3.3.3 and highlighted in blue text, has a far more powerful transmitter than the other class 1 devices used for this thesis. In general, Bluetooth is a short range technology designed to communicate up to distances of 10 metres. However, longer ranges are possible that cover far greater distances (up to 1 kilometre in perfect conditions using the more powerful 100 mW class 1 devices.

	Maximum Permitted Power		
Class	**(mW)**	**(dBm)**	**Typical Range**
Class 1	200 mW (Aircable Host XR)		>100 metres
Class 1	100 mW	20 dBm	~100 metres
Class 2	2.5 mW	4 dBm	~10 metres
Class 3	1 mW	0 dBm	~1 metre

Table 2: Bluetooth classes, power and typical ranges

Class 1 devices have the ability to increase or decrease their transmission power to the appropriate level based on the *Received Strength Signal Indictor* (RSSI) reading. This has implications when attempting to physically track a device. Class 2 and 3 devices do not have this capability, as they seek to conserve power and focus on shorter communication distances. In addition to the range of a device, the data transfer rate depends on which version of Bluetooth is supported on the particular device. Table 3 shows the different possible data transfer rates. Table 3 is particularly important, as some of the hardware described later does have limitations in the Bluetooth versions they support.

Bluetooth Version	**Data rate**	**High Data Rate Traffic**	**Release Year**
1.2	1 Mbit/s	721 Kbit/s	2003
2.0 +EDR	3 Mbit/s	>80 Kbit/s	2007
3.0 HS	24 Mbit/s	802.11 link	2009
4.0	24 Mbit/s	802.11 link	2013

Table 3: Bluetooth versions data transfer comparison

The data rates in Table 3 are not particularly high, the higher data rates cited in the table are only achieved by utilising WiFi, specifically the IEEE 802.11g physical layer (not Bluetooth).

Bluetooth packets start with a code that is based on the *Lower Address Part* (LAP) of a particular *Bluetooth Device Address* (BD_ADDR). The BD_ADDR is a 48 bit MAC address, just like the MAC address of an Ethernet device. The LAP consists of the lower 24 bits of the BD_ADDR and is the only part of the address that is transmitted with every packet (Ossmann, 2014). The BD_ADDR structure is described in more detail in Table 4 below. For example, with a Bluetooth device address of **11:22:33:44:55:66**, you only need to know **00:00:33:44:55:66** to communicate with the device. Since the *Upper Address Part* (UAP) is only 8 bits long, if you have the LAP, you will very quickly be able to interact with the device, as you only need 2^8 (256) at most guesses, before it is found. Finding the LAP is key in terms of security.

	NAP Non-significant Address Part	UAP Upper Address Part	LAP Lower Address Part
Bits	16 bits	8 bits	24 bits
Sample	**00:00**	**33**	**44:55:66**
		Error check based on UAP CRC also based on UAP	Access Code is derived from LAP
			Manufacturer ensures this part is unique

Table 4: Bluetooth Device Address structure (Davies, 2014)

According to Ossmann (2014), this process can be speeded up by prioritising common UAPs, possible due to the UAP being part of the *Organisationally Unique Identifier* (OUI) assigned to a relatively small number of manufacturers. These common OUIs can be identified thanks to the BNAP BNAP Project (Wright, 2015).

Poole (2007) defines two types of Bluetooth link that are available and can be set up; *Asynchronous Connection-oriented Logical* (ACL) and *Synchronous Connection Orientated* (SCO) communications links. ACL is the more widely used. Poole (2007) also categorises three main elements that are included in the higher layer stack or Bluetooth host; *Logical Link Control and Adaptation Protocol* (L2CAP), *Service Discovery Protocol* (SDP) and *Generic Access Protocol* (GAP). L2CAP is used to provide an interface for all the data requests that use the ACL links. The Bluetooth L2CAP affords multiplexing between the higher layer protocols which enables several applications to use the same lower layer links. SDP allows devices to discover which services other Bluetooth devices support, and list what the Bluetooth device supports. Bluetooth GAP describes how Bluetooth devices are able to discover each other and establish connections. It is one of the most basic Bluetooth profiles, but is used by every other profile as the foundation for establishing a link. Bluetooth GAP can put the device into three different modes of discovery; **General discovery, Limited discovery and Non-discoverable**. These discovery modes are an important theme for this thesis, and are discussed in more detail later.

For Bluetooth devices to converse correctly, Bluetooth Profiles are required. Bluetooth profiles are additional protocols that build upon the basic Bluetooth standard to more clearly define what kind of data a Bluetooth module is transmitting. While Bluetooth specifications explain how the technology works, profiles explain how it's used. A list of the thirty five most common Bluetooth Protocols is shown in Appendix A. Of particular note in this list, from a security point of view, is item 24 *OBject EXchange* (OBEX) and item 30 *Service Discovery Application Profile* (SDAP).

2.2.2 Bluetooth Low Energy

Originally called Wibree, *Bluetooth Low Energy* (BLE) technology was introduced in 2010, through the Bluetooth v4.0 specification. With its low power consumption and new features, BLE enables new applications that were impractical with Bluetooth Classic technology. BLE is an exciting and rapidly growing area of Bluetooth, providing functionality where low power may be a necessity, for example where a device is battery powered, but needs to be available for months or years.
The BLE standard offers a number of advantages over Bluetooth Classic, including low cost, low peak, average and idle mode power consumption, small in size making them useful for accessories and *Human Interface Devices* (HID).

BLE connections are quite simple, more so than the hop pattern of Bluetooth Basic Rate (BR) (Ryan, 2014). Table 5 focusses on these key differences between Bluetooth BR and *Enhanced Data Rate* (EDR) versus Low Energy. Two items of note from this table (highlighted in red) are the reduced number of channels, and the low maximum output power allowed (resulting in reduced ranges).

Characteristic	Bluetooth BR/EDR	Bluetooth LE
RF Physical Channels	79 channels, 1 MHz channel spacing	40 channels, 2 MHz channel spacing
Discovery/Connect	Inquiry/Paging	Advertising
Number of Piconet Slaves	7 (active)/255 (total)	Unlimited
Device Address Privacy	None	Private Device Addressing available
Max Data Rate	1-3 Mbps	1 Mbps via GFSK modulation
Encryption Algorithm	E0/SAFER+	AES-CCM
Typical Range	30 metres	50 metres
Max Output Power	100 mW (20dBm)	10mW (10 dBm)

Table 5: Key differences between Bluetooth BR/EDR and BLE (Padgette, et al., 2012)

2.2.3 Discoverable Mode

A device is said to be in discoverable mode when it periodically checks whether other devices are looking for them. Due to the sophisticated nature of Bluetooth technology, and specifically FHSS, Bluetooth creates its connections in a complicated manner. These come in the form of Master-Slave connections, these connections remain in place until they are broken, either by a disconnection, or by a poor quality link that makes communications impossible (i.e. the devices go out of range).

According to Walker (2012), Bluetooth devices have two modes: a discovery mode and a pairing mode. Discovery mode determines how the device reacts to inquiries from other devices looking to connect, and it has three actions. The discoverable action has the device respond to all inquiries, limited discoverable restricts that action, and non-discoverable tells the device to ignore all inquiries. It is a security risk to leave a device in discovery mode. Pairing should be controlled and mutual authentication should be practiced. It is never a good idea to respond to any request for pairing or PIN unless the user has initiated the pairing sequence. (Wireless Special Interest Group; PCI Security Standards Council, 2011).

Historically, Bluetooth security recommendations included turning off discoverable mode. The National Institute of Standards and Technology (NIST), highlights that discoverable devices are more prone to potential attack. Tipton (2012) notes that Bluetooth device owners may be unaware of their device's inherent vulnerabilities. Being able to retrieve a Bluetooth devices' BD_ADDR is all that is technically required to establish a connection with a remote device. Many devices rely on this secrecy of the BD_ADDR for security. In order to facilitate this, Bluetooth devices can be configured in discoverable mode, where they answer page request messages from other devices with their BD_ADDR information, and in non-discoverable mode, where they ignore requests for the BD_ADDR. "Turning off discoverability does nothing to thwart skilled attackers. Worse, it creates a false sense of security and makes it harder for the good guys" (Davies, 2013).

2.2.4 Non-Discoverable Mode

"It's likely that credit card skimming devices will be configured in non-discoverable mode if the bad guys want to evade detection" (Wright, 2011). According to Wright and Cache (2015), a device is said to be non-discoverable if it simply ignores (or doesn't look for) discovery requests, they go on to say that many devices aren't discoverable by default, so you must enable this feature specifically, usually for a brief period of time.

Keeping a device in non-discoverable mode is a standard security practice, but is not a security fix.
Unlike IEEE 802.11, Bluetooth does not transmit the full BD_ADDR, which makes it possible to capture the last three bytes of the BD_ADDR (LAP). Once these three bytes are known, a user can send connection request messages to every common BD_ADDR prefix, or OUI until, the full BD_ADDR found.

In other words, the most important passive Bluetooth monitoring function is simply capturing the LAP from each packet transmitted on a channel. LAP sniffing allows you to identify Bluetooth devices operating in your vicinity. A hardware limitation, until recently was this inadequacy of Bluetooth devices, For example, the Azio Bluetooth adaptor highlighted in Figure 1, is an active device and can only discover devices that have discovery mode enabled. This particular device, has little value working in proximity to devices in non-discoverable mode. However, this specific drawback/constraint is addressed by the Ubertooth One, which is described in more detail in later sections.

Figure 1: Azio BTD-V201 USB Micro Bluetooth Adapter, Class 1

2.2.5 Bluetooth Security Issues

Bluetooth is a pervasive technology, according to Padgette (2012), Bluetooth technology has been integrated into many types of business and consumer devices, including cell phones. Laptops, automobiles, medical devices, printers, keyboards, mice and headsets. This can lead to problems, because security setting will be different on each device, making it difficult to follow generic security advice.

Due to the large volume of devices, and a plethora of device types, Bluetooth security is a big issue. Many devices are vulnerable to an excess of attacks included denial of service (DOS), main-in-the-middle (MITM) attacks, eavesdropping, etc. Some of the more common Bluetooth attacks/vulnerabilities, are presented in Table 6.

Attack Name	Description
Bluebug attack	"An attacker can use the AT commands on a victim's cell phone to initiate calls, send SMS messages, etc." (Tipton, 2010). "This form of Bluetooth security issue allows hackers to remotely access a phone and use its features. This may include placing calls and sending text messages" (Poole, 2007)
Bluejacking	"Allows an anonymous message to be displayed on the victim's device." (Tipton, 2010). "Often, the Bluejacker is trying to send someone else their business card, which will be added to the victim's contact list in their address book" (Harris, 2013)
Bluescarfing	"Bluescarfing is the actual theft of data from a mobile device" (Walker, 2012)
Bluesnarfing	"Bluesnarfing is the unauthorised access from a wireless device through a Bluetooth connection. This allows access to a calendar, contact list, e-mails, and text messages, and on some phones users can copy pictures and private videos." (Harris, 2013)
Bluesmacking	"Bluesmacking is simply a denial-of-service attack against a device." (Walker, 2012)
Bluesniffing	"Bluesniffing is exactly what it sounds like" (Walker, 2012)
Buffer overflow	"Buffer overflow: An attacker can remotely exploit bugs in the software on Bluetooth-enabled devices." (Tipton, 2010)
Car Whispering	"Car Whispering: This involves the use of software that allows hackers to send and receive audio to and from a Bluetooth enabled car stereo system" (Poole, 2007)

Table 6: Common Bluetooth attacks/vulnerabilities

This all leads to difficulties for the individual with responsibility for the Bluetooth region of the attack landscape. Without a means to monitor active Bluetooth devices, nor having the capability to passively sniff those device's traffic, it is difficult to determine if any attacks took place, eavesdropping traffic for example.

2.3 Intrusion Detection

An *Intrusion Detection System* (IDS) is a system used to determine whether unauthorised access (intrusions) are occurring on a network. Once identified, mitigating steps can be initiated, perhaps using an *Intrusion Prevention System* (IPS). For this thesis no prevention will take place, but the monitoring of local Bluetooth traffic does take place. The Wireless SIG (2011) assert that intrusion detection is the process of monitoring the events occurring in a computer system or network and analysing them for signs of possible incidents which are violations or imminent threats of violation of security policies. They go on to say that an IDS is the actual software that automates the process. An IPS is a system that has all the capabilities of an IDS but can also attempt to stop possible incidents.

IDS/IPS are well established in Wi-Fi (802.11) but are limited in their Bluetooth (802.15) support, because of the volume of such devices and the nature of the technology. There are three main categories of IDS, *Network-based Intrusion Detection System* (NIDS), *Host-based Intrusion Detection System* (HIDS) and *Distributed Intrusion Detection System* (DIDS)

The toolkit described later is neither a NIDS based system, nor a HIDS based system, but can be considered a DIDS type system sensor. “The sensor can function in promiscuous mode or non-promiscuous mode. However, in all cases, the DIDS’s defining feature requires that the distributed sensors report to a central management station” (Kohlenberg, et al., IDS and IPS Toolkit). While no central management is described below, it is the feasibility of the sensor itself that is being tested.

If a successful sensor combination is found, it can be added to existing security toolsets. Koziol (2003) tells us that an IDS is a critical component in a defence-in-depth information security strategy. Defence in depth is the method of protecting information resources with a series of overlapping defensive mechanisms. The idea being if one defence fails, others will thwart an attack. These systems are difficult to implement, with false positives being very common in early stages while issues are being ironed out. Logging devices is very important. In combination with an up to date hardware inventory a wireless IDS/IPS should be able to observe all APs and clients, on all operational channels, and classify each device as authorised, unauthorised/rogue or neighbouring. A SYSLOG type system would be useful for this purpose.

2.4 Compliance and Guidelines

This thesis investigates how this use of wireless technologies, and Bluetooth in particular, can affect the overall compliance landscape of an organisation, with particular emphasis on the standards used by the *Payment Card Industry Data Security Standard* (PCI-DSS), who require that organisations regularly assess their networks for these rogue AP threats, and many vendors have implemented products designed to address this threat.

2.4.1 PCI-DSS

Created by four credit card companies in 2004; Visa, MasterCard, American Express and Discover; the PCI-DSS provides a minimum set of requirements created by the PCI Security Standards Council. The purpose of these standards is to help protect credit card data. The full specifications of PCI-DSS are available at the PCI Standards Security Council website, and are summarised in Table 7 below, which highlights the six main objectives and twelve requirements. It should be noted that these are minimum requirements. Being PCI compliant does no meant that the data is completely safe from attack.

PCI –DSS Objectives	PCI-DSS Requirements
1. Build and Maintain a Secure Network and Systems	1. Install and maintain a firewall configuration to protect cardholder data
	2. Do not use vendor-supplied defaults for system passwords and other security parameters
2. Protect Cardholder Data	3. Protect stored cardholder data
	4. Encrypt transmission of cardholder data across open, public networks
3. Maintain a Vulnerability Management Program	5. Protect all systems against malware and regularly update anti-virus software or programs
	6. Develop and maintain secure systems and applications
4. Implement Strong Access Control Measures	7. Restrict access to cardholder data by business need to know
	8. Identify and authenticate access to system components
	9. Restrict physical access to cardholder data
5. Regularly Monitor and Test Networks	10. Track and monitor all access to network resources and cardholder data
	11. Regularly test security systems and processes
6. Maintain an Information Security Policy	12. Maintain a policy that addresses information security for all personnel

Table 7: PCI-DSS Objectives and Requirements

Several of these requirements are affected by the inherent weaknesses of Bluetooth, and they do require a lot of work to implement. Cisco (2014) advises maintaining the physical security of wireless data, and having a person at each physical location responsible for checking if equipment has been tampered with or compromised in any way. This person must manually assess (utilising vendor guidance) the security of the access points, wireless controllers, and any other physical pieces of the organisation's WLAN.

The PCI-DSS Security Standards Council (2011) recommends periodic detection and identification of unknown and potentially dangerous rogue wireless devices, as well as documented response procedures in the event unauthorised wireless devices are detected. Which of course is particularly difficult for Bluetooth.

In order for effective detection to take place, it is vital that an updated hardware inventory, including Bluetooth devices (BD_ADDR and friendly device name information), be constantly updated and maintained. This is important so legitimate devices can be distinguished from illegitimate devices. Besides actively scanning the ISM band, physical and manual inspections of APs, hardware and networking devices is also important, as it may indicate whether unauthorised devices are connected or not. This physical inspection, will not however, tell an auditor if devices had been connected in the past, and subsequently removed.

When a rogue device is discovered, it then needs to be logged, and/or disabled. A verification scan could also be run. Of relevance to Bluetooth are requirements 11 and 12. The standards indicate that rogue threats need to be immediately resolved, with the environment rescanned as soon as possible.

Protecting the data in its own environment is of paramount importance to the PCI-DSS standards, where it is categorised as the *Cardholder Data Environment* (CDE), and is comprised of people, processes, and technology that store, process, or transmit cardholder data or sensitive authentication data. The PCI-DSS specification is very specific in its definition of CDE, and how it comes into scope, or not, for a wireless (Bluetooth) network co be out of scope from a PCI audit, it must be completely isolated from the CDE, with no possibility of traffic between the two environments.

The PCI-DSS wireless Special Interest Group offer some specific Bluetooth recommendations summarised in Table 8. However, in relation to item 2, as previously stated, turning a device to undiscoverable is now no longer an effective defence, due to the passive scanning ability of the Ubertooth One. The author believes this recommendation could be updated to state explicitly that a better defence would be to turn Bluetooth off, unless required, and then only turned on when needed.

Bluetooth Configuration Recommendations
1. Choose PIN codes that are sufficiently random and long. Avoid static and weak PINs, such as all zeroes.
2. Bluetooth devices should be configured by default as, and remain, undiscoverable except as needed for pairing.
3. Ensure that link keys are based on combination keys rather than unit keys. Do not use unit keys.
4. For v2.1 devices using Secure Simple Pairing, do not use the —Just Works model.
5. Perform service and profile lockdown of device Bluetooth stacks. Do not allow the use of multiple profiles in the unit.
6. In the event a Bluetooth device is lost or stolen, immediately unpair the missing device from all other Bluetooth devices with which it was previously paired.

Table 8: PCI-DSS Bluetooth recommendations

2.4.2 NIST Special Publications

Besides the direction given by the Payment Card Industry, the National Institute of Standards and Technology (NIST) also provide several *Special Publications* (SP) in the 800 series, which are of particular interest to the computer security community. The three most relevant publications for this thesis are described below:

1. **NIST SP 800-121 (Revision 1) – Guide to Bluetooth Security** (2012), which supersedes NIST SP 800-121 – Guide to Bluetooth Security (2008). Provides excellent guidance from experts in the field. Besides the authors, acknowledgements include Michael Ossmann of Great Scott Gadgets (and creator of the Ubertooth One), and David Trainor of Cambridge Silicon Radio Ltd. In general this proved an excellent source of information, however no mention of the Ubertooth is made for example, and passive eavesdropping is only mentioned in the context of device pairing. This document does offer a thorough Bluetooth Mitigation Checklist of 33 items, which is transcribed into Appendix B. Nevertheless, with regards to the checklist itself, for item 6, the author believes that maintaining a complete checklist should be a recommended practice. This checklist would enable an auditor to identify rogue devices, while helping to trace the origin of these rogue devices.

   For item 33, designating an individual to track the progress of security Bluetooth products, should also be a recommended practice in this author's opinion, as different threats are being created, and different vulnerabilities are being exploited, as highlighted in Section 2.2.5. As these issues are identified, they can be tracked and addressed by an individual with the right subject matter expertise.

Item 16 in the list – Bluetooth devices should be configured by default as undiscoverable and remain undiscoverable except as needed for pairing, is particularly relevant to this thesis, as a passive sniffing device similar to the Ubertooth One does not care whether this setting is turned on or not.

2. **NIST SP 800-124 (Revision 1) – Guidelines for Managing the Security of Mobile Devices in the Enterprise** (2013), which supersedes NIST SP 800-124 – Guidelines on Cell Phone and PDA Security (2008). This document is high level, and offers advice on policies to manage mobile devices in the enterprise. Some of the more general advice offered around Bluetooth include limiting user access and application access to hardware devices, including Bluetooth, while actively managing wireless interfaces (Bluetooth and WiFi for example).
3. **NIST SP 800-94 – Guide to Intrusion Detections And Prevention Systems (IDPS)** (2007). This publication offers some basic guidance on wireless Intrusion Detection outlining how it is most commonly deployed within range of an organisation's wireless network to monitor it, but also can be deployed to locations where unauthorised wireless networking could be occurring. This document unfortunately has no literature covering Bluetooth specifically, and actually says as such, but does provide important advice for WiFi that may be transferrable, such as recommendations on sensor locations, recommending that wireless sensors actively monitor *Radio Frequency* (RF) ranges used by the organisation. Also offered is a valuable guide to data fields that should be logged by such devices, as depicted in Table 9, which would be helpful for developing SYSLOG type functionality, for example.

Data fields for Wireless IDS logging
Timestamp (usually date and time)
Event or Alert type
Priority or severity rating
Source MAC address (the vendor is often identified from the address)
Channel number
ID of the sensor that observed the event
Prevention action performed (if any)

Table 9: Data fields for Wireless IDS logging (Scarfone & Mell, 2007).

2.4.3 Compliance – Overlap with WiFi

Bluetooth and WiFi have a lot in common, including the sharing of the same ISM range. Nardi (2012) outlined three distinct steps required, common to both technologies, used to collect all of the pertinent data for each device. Step one is to command the hardware to scan all available channels for discoverable devices and return their MAC addresses. Step two was to list the MAC addresses gathered, each device could then be queried to determine the device's human friendly name. Finally, with the device's MAC and name recorded, the system can then use Service Discovery Protocol to find out the high-level services the target device offers

According to Poole (2007), one of the disadvantages of the original version of Bluetooth was that the data rate was not sufficiently high, when compared to other wireless technologies such as 802.11. In November 2004, a new version of Bluetooth, known as Bluetooth 2 was ratified which delivered enhanced speeds (EDR) increasing the maximum data rate to 3Mbps, a significant increase on what was available in the previous Bluetooth specifications.

However, it wasn't until Bluetooth Version 3, until aspects of both technologies merged. Bluetooth 3 enables these much higher speeds by utilising a collocated IEEE 802.11 link, the Bluetooth link being used for the negotiation and establishment of the WiFi connection.

Even though Bluetooth was now using 802.11 technology to enjoy higher speeds, it lacked the generic wireless sniffing tools that generally available in the WiFi arena. At present, many WiFi devices have the capability to monitor, and tamper with, wireless networks. Until recently, this capability was not cheaply available with Bluetooth devices according to Peter (2011), if it was available, an auditor/penetration testers could actively monitor the Bluetooth spectrum.

However, the Ubertooth One, described more fully in the next chapter, now makes it possible for this active monitoring to take place. One thing that sets the Ubertooth apart from other Bluetooth platforms is its capability of not only sending and receiving 2.4 GHz signals, but also operating in monitor mode, monitoring Bluetooth traffic in real-time. This mode has been available in commodity WiFi modules for years and has found myriad uses in research, development and security auditing but no such solution existed for the Bluetooth standard until now.

CHAPTER 3 - Software and Hardware Requirements

3.1 Introduction

According to the PCI Wireless Special Interest Group (2011) wireless analyser range from free tools to more expensive commercial scanners, whose purpose is to sniff for wireless devices within the vicinity and identify them. By doing this an auditor can audit a site, and then manually investigate rogue signatures to determine if the device has access to the CDE or not. In this way devices could be classified as rogue, authorised or a neighbouring device.

While this works for WiFi for Bluetooth a new toolset is required. This chapter introduces off-the-shelf hardware capable of performing the described functions, and presents some of the required software. In particular, the **hciconfig** tool is used to describe attributes of the hardware devices used. For the hardware section, a brief description of the features of each device is given, starting with the Ubertooth one, and introducing a number of commodity Bluetooth devices, with a view to discovering the best Ubertooth combination, highlighting each device's benefits and limitations. Also outlined are reasons why some particular devices were chosen for further testing, while others discarded.

3.2 Software Requirements

A brief description of the main operating system used for this thesis, Kali Linux is detailed below. "BlueZ is a powerful Bluetooth communications stack with extensive APIs that allow a user to fully exploit all local Bluetooth resources. It is open source, freely available, and comes with all major distributions of GNU/Linux" (Huang & Rudolph, 2007, p. 67). BlueZ commands used in testing are described in more detail in the following sections. Finally, an explanation of the function and use of the ubertooth-scan is given.

3.2.1 Kali Linux

A Kali Linux 1.09a distribution was downloaded for use on the laptop. The Ubertooth One was developed for use primarily on a Linux distribution. Kali Linux displaces Backtrack, is based off Debian, and designed specifically for penetration testing. An ISO file is an archived file of an image. The 64-bit version of Kali was an ISO file that was 2.84 GB in size. In order for Kali to be installed on the Thinkpad laptop, the ISO had to be expanded out. A program called Win32DiskImager was used for this purpose, enabling the Kali distribution installation files to be loaded onto a USB memory stick.

Installing Kali was a straightforward process, as the hard drive was a new one, and there was no need to create a dual-bootable device.

3.2.2 BlueZ

There are three parts to a Bluetooth subsystem on Linux, the kernel routines, the libbluetooth library, and the six user tools. The user tools "are indispensable when configuring or modifying Bluetooth devices on a machine and debugging applications" (Huang & Rudolph, 2007, p. 182). These commands are briefly described in Table 10, with **hciconfig** and **hcitool** being described in more detail in the following sections. These commands are extremely useful, and describe in-depth details of attached Bluetooth devices.

Tool Name	Tool Description
`hciconfig`	Configure the basic properties of local adapters
`hcitool`	Detect nearby devices; display information on and adjust low-level connections
`sdptool`	Search for and browse SDP services. Basic configuration of locally advertised services
`hcidump`	Low-level debugging of connection setup and data traffic
`l1ping`	Test L2CAP connection functionality
`uuidgen`	Generates unique UUID for use with SDP

Table 10: Bluetooth Linux Tools Quick Reference (Huang & Rudolph, 2007, p. 180)

3.2.3 BlueZ – hciconfig

“The command hciconfig is used to configure the basic properties of Bluetooth adapters. As the name suggests, it provides a user-level interface to the (HCI) protocol. When invoked without any arguments, it will display the status of the adapters attached to the local machine” (Huang & Rudolph, 2007, p. 182).

By running the **hciconfig** command without any options, the connected devices are displayed, an example is shown in Appendix C. It is very important to note that while the all devices are displayed, their order appears random, it is not ordered according to HCI number, nor based on BD_ADDR.

As shown in Appendices E to K, the command options used to gain information about devices used were: **hciconfig hciX –a** and **hciconfig hciX revision**, the former **–a** displays basic info, print features, packet type, link policy, link mode, name, class, version. The latter revision displays revision information, which displays firmware information about the device (for example, BlueCore4 with external firmware EEPROM, as shown in Appendix J for the AIRcable Host XR). Besides displaying important information about each attached device, this command can be used to enable/disable devices. For example to disable device hci1, the command **hciconfig hci1 down** would be used. To enable a device the up command would be used, for example; **hciconfig hci1 up**.

Note that hciconfig changes are temporary, so any changes made will be lost if the device is rebooted.

3.2.4 BlueZ - hcitool

The **hcitool can** be used for Bluetooth discovery and basic enumeration. When scanning, hcitool caches information about devices, reporting the presence of devices that were once observed but may no longer be in range. By default, hcitool shows only BD_ADDR and device name information, but can collect additional details by adding the **all** parameter. This tool has two main uses. Firstly to search and detect nearby Bluetooth devices, secondly to test and show information about low-level Bluetooth connections. “In a sense, hcitool picks up where hciconfig ends – once the Bluetooth Adapter starts communicating with other Bluetooth devices” (Huang & Rudolph, 2007, p. 185).

It should be noted that the hcitool should be run with root privileges, otherwise only a limited amount of information will be returned about the device being polled.

3.2.5 ubertooth-scan

In order to set up the Ubertooth with the necessary software, two sets of companion instructions were sourced and followed from Pentura Lab's Blog; **Ubertooth – Open-Source Bluetooth Sniffing** (Davies, 2013) and **Ubertooth – Bluetooth Sniffing Updated for 2014!** (Davies, 2014)**.** As Kali is branched off Debian Linux, the Debian specific instructions were followed, and the Ubertooth components were successfully installed.

With the Ubertooth configured, low-level Bluetooth data can be captured to identify non-discoverable devices in the area. To do so, Wright and Cache (2015) describe running the **ubertooth-rx** utility to discovery these LAPs of active Bluetooth devices. Davies (2014), noted that ubertooth-rx replaces the deprecated commands (lap, uap, hop).

ubertooth-scan takes this one level further, "**ubertooth-scan** uses the LAP recovery feature of ubertooth-rx with an Ubertooth interface, but it also uses the Linux BlueZ Bluetooth interface with a traditional Linux dongle to validate a potential NAP *[sic]* for the identified LAP. In this fashion, ubertooth-scan speeds up NAP *[sic]* recovery while eliminating false-positives" (Wright & Cache, 2015, p. 219).

Ubertooth-scan requires both an Ubertooth and a standard Bluetooth device on a host with BlueZ installed. The tool uses the Ubertooth to passively sniff for Bluetooth packets, retrieving the LAP (and eventually) UAP values before handing them to libbluetooth to query the device name.

ubertooth-scan was the primary survey tool used for this thesis. Described in Table 11 are sample outputs from the three different kinds of scans that ubertooth-scan tool is capable of delivering.

		Command Features				
Command	**Description**	**Initiate Standard Device Scan**	**hcitool Type Scan**	**Check for Supporting Features**	**Chipset Version**	**Clock Offset**
`ubertooth-scan`	Basic Scan	✓				
`ubertooth-scan -s`	HCI Type Scan	✓	✓			
`ubertooth-scan -x`	Extended Scan	✓		✓	✓	✓

Table 11: ubertooth-scan options explanation

The LAP of nearby devices appeared to be quite easy to find in each case and this was clearly seen in all three scan types, as they were capable of finding LAPs and their corresponding UAPs. The basic scan appears to ignore discoverable devices, and concentrates only on the non-discoverable ones for LAP capture and UAP enumeration.

The HCI type scan does not actually use the BlueZ hcitool, but it does call the same library functions (libbluetooth), and performs the equivalent of running the command: **hcitool scan** to return their full Bluetooth Device Address and the device friendly name. The devices shown for this part of the scan are discoverable devices. The HCI type scan then continues with capturing non-discoverable LAPs and attempting to discover their corresponding UAPs.

As noted by Spill (2015) the extended scan uses the Ubertooth One to find devices that are transmitting within range, prior to offloading to your Bluetooth device to perform the extended scan on the devices found. 4 bytes of the BD_ADDR address are required for this to work.

This command should produce output similar to running the BlueZ command: **hcitool info**. It is possible to run both the hci type scan, and the extended scan at the same time. A copy of the output produced from **ubertooth-scan** help can be found in Appendix D.

Truncated outputs from running each scan type is displayed in Table 12 below. Included are lines that refer to the *Adaptive Frequency Hopping* (AFH) map, which allows Bluetooth connections to avoid using noisy channels, for example, nearby wireless networks. The map stipulates which channels are obtainable for a given connection.

Command: ubertooth-scan -b hci3 -t 900

```
Ubertooth scan
systime=1425988306 ch=20 LAP=9db2cd err=2 clk100ns=880354654 clk1=13772345 s=-81 n=-86 snr=5
systime=1425988336 ch=10 LAP=968e95 err=0 clk100ns=1171628600 clk1=13818948 s=-58 n=-84 snr=26
systime=1425988348 ch=77 LAP=968e95 err=1 clk100ns=1296829853 clk1=13838981 s=-73 n=-89 snr=16
We have a winner! UAP = 0xc8 found after 2 total packets.
systime=1425988361 ch=25 LAP=968e95 err=0 clk100ns=1425996238 clk1=13859647 s=-54 n=-86 snr=32
systime=1425988366 ch=14 LAP=166ff6 err=2 clk100ns=1480119752 clk1=13868307 s=-69 n=-83 snr=14
systime=1425988367 ch=42 LAP=968e95 err=0 clk100ns=1490858100 clk1=13870025 s=-63 n=-85 snr=22
                                 [entries removed for brevity]
00:00:00:9D:B2:CD
         AFH Map=0x00000000000000100000
00:00:C8:96:8E:95  [unknown]
         AFH Map=0x200000000ffb6e056628
00:00:00:16:6F:F6
         AFH Map=0x00000000000000004000
                                 [entries removed for brevity]
```

Command: ubertooth-scan -b hci3 -t 900 -s

```
HCI scan
DC:9F:A4:35:D3:F9  Nokia C5
B0:89:91:9F:3B:47  JimmyJohnJoe
7C:6D:62:95:2F:EE  PR2262-macJC
7C:6D:62:95:33:44  PR2279-macMD

Ubertooth scan
systime=1425989258 ch=25 LAP=968e95 err=1 clk100ns=568313735 clk1=15295282 s=-54 n=-86 snr=32
systime=1425989281 ch= 3 LAP=2851e7 err=2 clk100ns=795228256 clk1=15331588 s=-63 n=-81 snr=18
systime=1425989285 ch=71 LAP=01fc5b err=2 clk100ns=833615898 clk1=15337730 s=-88 n=-89 snr=1
systime=1425989293 ch= 0 LAP=f113fb err=2 clk100ns=913482935 clk1=15350509 s=-75 n=-83 snr=8
systime=1425989294 ch=61 LAP=968e95 err=0 clk100ns=927407147 clk1=15352737 s=-63 n=-82 snr=19
systime=1425989297 ch=44 LAP=968e95 err=1 clk100ns=957335144 clk1=15357525 s=-63 n=-87 snr=24
systime=1425989297 ch=44 LAP=968e95 err=1 clk100ns=957339144 clk1=15357526 s=-63 n=-87 snr=24
                                 [entries removed for brevity]
00:00:00:96:8E:95
         AFH Map=0x4b802000d5400a040092
00:00:00:28:51:E7
         AFH Map=0x00000000000000000008
00:00:00:01:FC:5B
         AFH Map=0x00800000000000000000
00:00:00:F1:13:FB
         AFH Map=0x00000000000000000001
                                 [entries removed for brevity]
```

Command: ubertooth-scan -b hci2 -t 900 -x

```
Ubertooth scan
systime=1425987398 ch= 7 LAP=968e95 err=2 clk100ns=1625936341 clk1=12318774 s=-57 n=-83 snr=26
systime=1425987398 ch=18 LAP=968e95 err=1 clk100ns=1627097580 clk1=12318959 s=-64 n=-85 snr=21
systime=1425987398 ch=21 LAP=968e95 err=0 clk100ns=1627396992 clk1=12319007 s=-63 n=-86 snr=23
systime=1425987398 ch=23 LAP=968e95 err=0 clk100ns=1627599262 clk1=12319040 s=-61 n=-87 snr=26
We have a winner! UAP = 0xe3 found after 4 total packets.
systime=1425987398 ch=29 LAP=968e95 err=0 clk100ns=1628186223 clk1=12319134 s=-63 n=-86 snr=23
                                 [entries removed for brevity]
00:00:E3:96:8E:95  [unknown]
Requesting information ...
         AFH Map=0x3f844000777beff6ffff
00:00:00:7C:A3:68
         AFH Map=0x00000000000000040000
                                 [entries removed for brevity]
```

Table 12: ubertooth-scan sample outputs (heavily truncated)

3.3 Hardware Requirements

Sourcing capable Bluetooth hardware proved to be a more difficult task than initially anticipated. Nardi (2012) notes that some manufacturers incorrectly label their devices as Class 1, when they are not. Some other manufacturers, to save money, also duplicate BD_ADDR device addresses, rather than provide unique addresses. Two cheap USB devices were sourced. As Figure 2 shows, externally they were externally identical (apart from a few scratches) so the author marked one with a white dot to make them distinguishable.

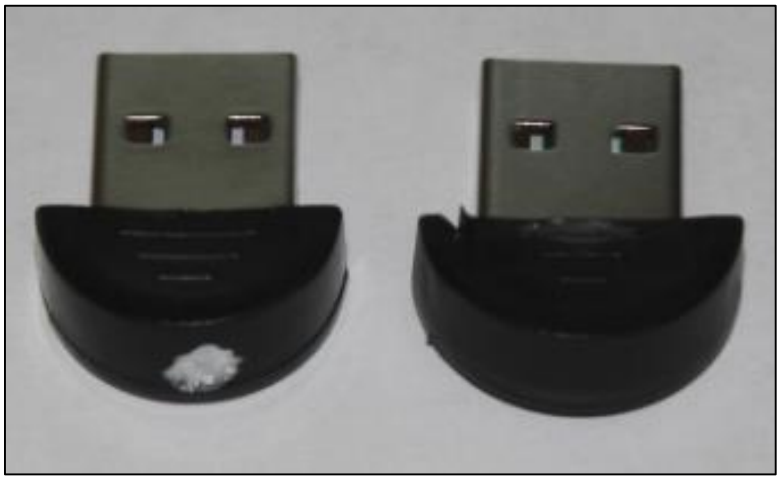
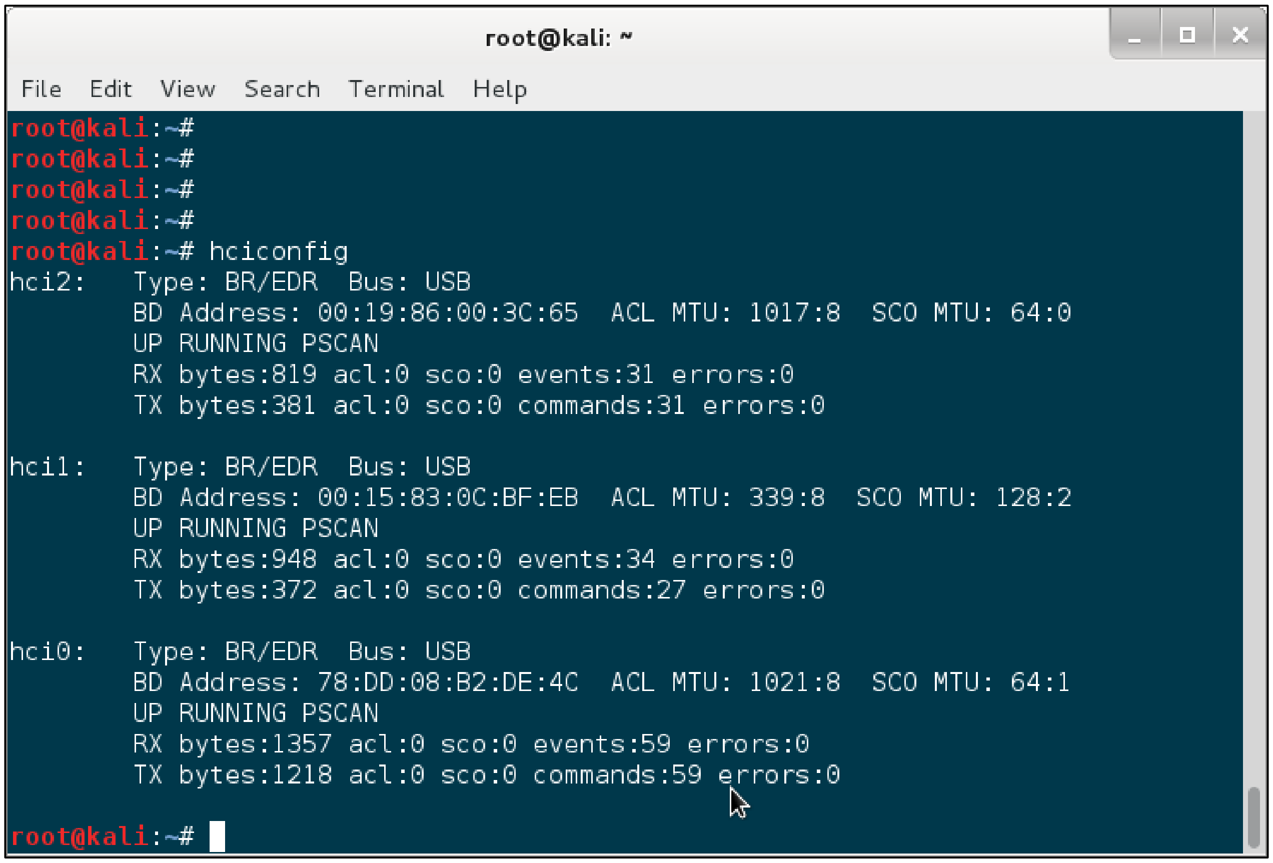


Figure 2: USB Nano devices

The Nano devices were then place into the Linux laptop, and the **hciconfig** command was run, which is shown in Figure 3. This showed that the devices were successfully picked up by the kernel, were running, and displayed BD_ADDR information. Note, that at this point in time, all three devices were up and running simultaneously.



```
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~# hciconfig
hci2:   Type: BR/EDR  Bus: USB
        BD Address: 00:19:86:00:3C:65  ACL MTU: 1017:8  SCO MTU: 64:0
        UP RUNNING PSCAN
        RX bytes:819 acl:0 sco:0 events:31 errors:0
        TX bytes:381 acl:0 sco:0 commands:31 errors:0

hci1:   Type: BR/EDR  Bus: USB
        BD Address: 00:15:83:0C:BF:EB  ACL MTU: 339:8  SCO MTU: 128:2
        UP RUNNING PSCAN
        RX bytes:948 acl:0 sco:0 events:34 errors:0
        TX bytes:372 acl:0 sco:0 commands:27 errors:0

hci0:   Type: BR/EDR  Bus: USB
        BD Address: 78:DD:08:B2:DE:4C  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN
        RX bytes:1357 acl:0 sco:0 events:59 errors:0
        TX bytes:1218 acl:0 sco:0 commands:59 errors:0

root@kali:~#
```

Figure 3: Running the hciconfig command in Linux

The terminal information captured above is summarised in Table 13 below. It should be noted, that the vendor column in the table above was discovered using the first 6 characters of the BD_ADDR (24 bits), and submitting a query at the **mac find** website (Coffer, 2013). However, the full list of device addresses is also available on the IEEE Standards Association's **Organisationally Unique Identifier** webpage for manual lookups.

Device Name	Bluetooth Device Address	Vendor
Marked Nano Device	BD_ADDR: **00:19:86:00:3C:65**	Cheng Hongjian
Unmarked Nano Device (LED)	BD_ADDR: **00:15:83:0C:BF:EB**	IVT corporation
Internal Device (Thinkpad built-in)	BD_ADDR: **78:DD:08:B2:DE:4C**	Hon Hai Precision Ind. Co.,Ltd.

Table 13: USB Nano devices summary

Even though the devices had different BD_ADDR addresses, the unmarked device proved unreliable, it was only picked up by the laptop after it was dismantled and reassembled, as shown in Figure 4 below. The indicator LED came on at this point indicating that the device was indeed working.

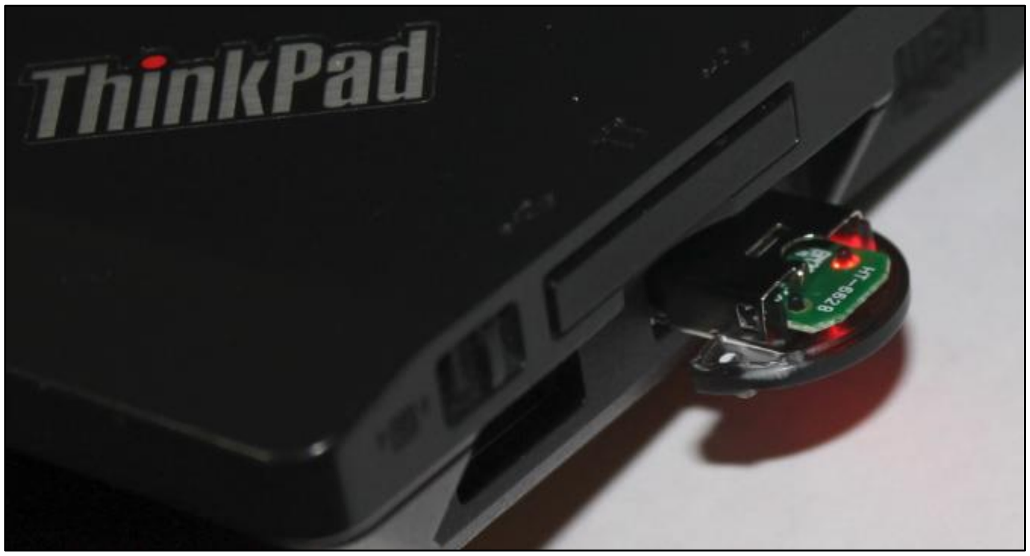


Figure 4: Intermittent working of Bluetooth LED

As shown in Figure 5, the zigzag lines near the top of the printed circuit board (PCB), are the antennas used by these Bluetooth dongles, it is more distinguishable in the unmarked device on the right. While it may be possible to attach an external antenna, these devices are not class 1, so lack significant power output, the benefits would be minimal.

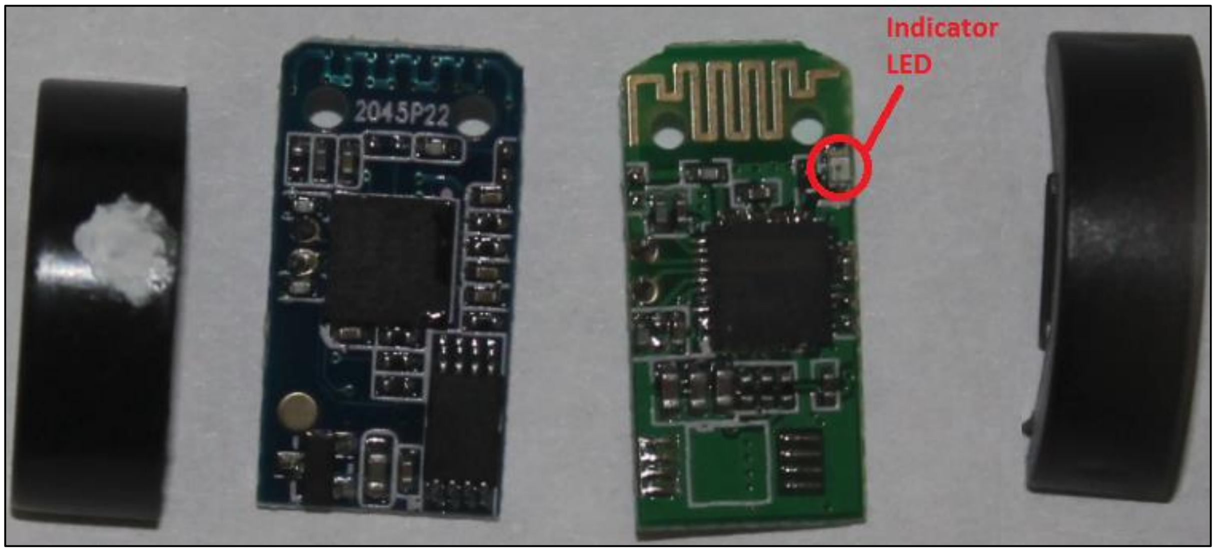


Figure 5: USB Nano hardware differences (white dot device on left, unmarked device on right)

The unmarked device - Nano USB Bluetooth (LED), was advertised with the specifications as highlighted in Table 14 below.

Nano USB Bluetooth Specifications
• Bluetooth v2.0 and v1.2 Compliant • Internal Red LED to show that the device is working correctly • Supporting profiles : Networking, Dial-up, Fax, LAN Access and Headsets • USB Interface • Symbol rate : 3 Mbps • Range : 20m

Table 14: Unmarked device advertised specifications

However, from running the **hciconfig -a** command, as shown in Appendix G. It was noted that this device was running with: `HCI Version: 2.0.`

According to the Bluetooth SIG Host Controller Interface webpage (Bluetooth SIG, 2015), a value of 2 indicates that Bluetooth Core Specification v1.2 is used, and NOT Bluetooth v2.0 as indicated in the specifications table above. Additionally, the Bluetooth SIG Link Manager Protocol (LMP) webpage (Bluetooth SIG, 2015) also shown in Appendix G with a value of: `LMP Version: 2.0`, confirms that Bluetooth Core Specification v1.2 is used.

Nardi (2010) recommends awareness of the Bluetooth chipset when purchasing devices, noting that the best supported and documented is the *Cambridge Silicon Radio* (CSR) chipset. This chipset has the advantage of tools and firmware modifications being readily available for it, which would be useful for penetration testing for example. The **hciconfig** command, showed that CSR was used the unmarked device (LED): `Manufacturer: Cambridge Silicon Radio (10).` However, due to the unreliability of this device its further use was discontinued.

The marked Nano device (Appendix F) and the laptop's internal device (see Appendix E) were neither based on CSR chipset, but based on the Broadcom chipset, because of this, the marked device was discarded from further use, as the internal device could be used for testing purposes to the same effect.

3.3.1 Ubertooth One

At Toorcon in 2010, Michael Ossmann presented Ubertooth Zero, which was the first prototype of Project Ubertooth. Prior to this, sniffing Bluetooth in a similar fashion to WiFi was both difficult and expensive. Regular Bluetooth dongles just did not have the necessary passive scanning functionality, Holeman (2013) highlighted that active Bluetooth scanning could be performed using commodity Bluetooth devices, however passive scanning required specialist hardware and software libraries. The Ubertooth Zero was such a specialist hardware and software product.

In 2011 at ShmooCon, Ossmann presented the Ubertooth One. Until then, expensive industrial equipment, or specialised *software defined radios* (SDR) were the only option to sniff Bluetooth packets. The Ubertooth One, was an off-the-shelf product, available at the hakshop website (Ossman, 2015) for just $100. The Ubertooth one had a power rating comparable to a Class 1 Bluetooth device. Shown in Figure 6 is the reverse side of the *printed circuit board* (PCB) of the Ubertooth One, the device is Open Source and employs a *Joint Test Action Group* (JTAG) interface for hardware debugging purposes.

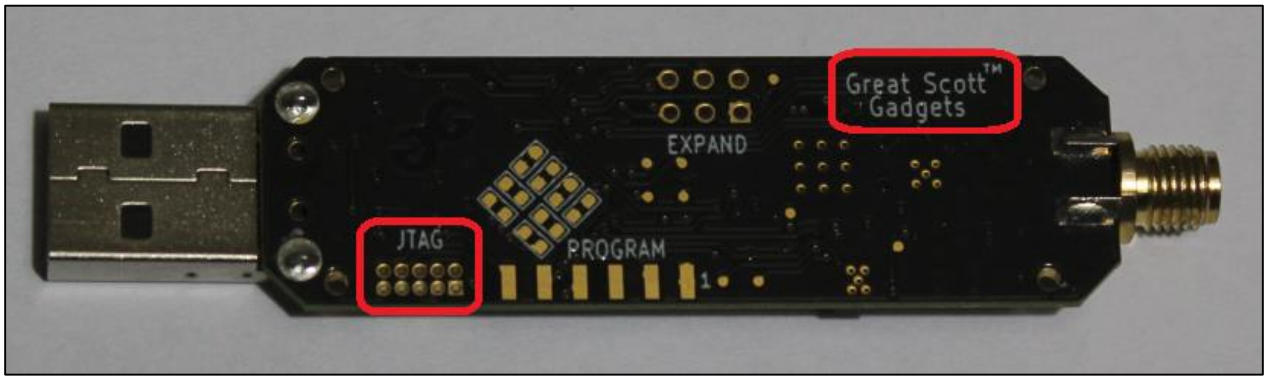


Figure 6: Ubertooth One, from Great Scott Gadgets, note the JTAG pins

Some of the main features of the Ubertooth One are described in Table 15 below.

Ubertooth One Features (Ossmann, 2013):
• 2.4 GHz transmit and receive. • Transmit power and receive sensitivity comparable to a Class 1 Bluetooth device. • Standard Coretex Debug Connector (10-pin 50-mil JTAG). • In-System Programming (ISP) serial connector. • Expansion connector: intended for inter-Ubertooth communication or other future uses. • Six indicator LEDs.

Table 15: Ubertooth One features

Shown in Figure 7 is the operational side of the Ubertooth One PCB. The key components of the device are outlined and highlighted, including the various indicator Light Emitting Diodes (LEDs) that are used.

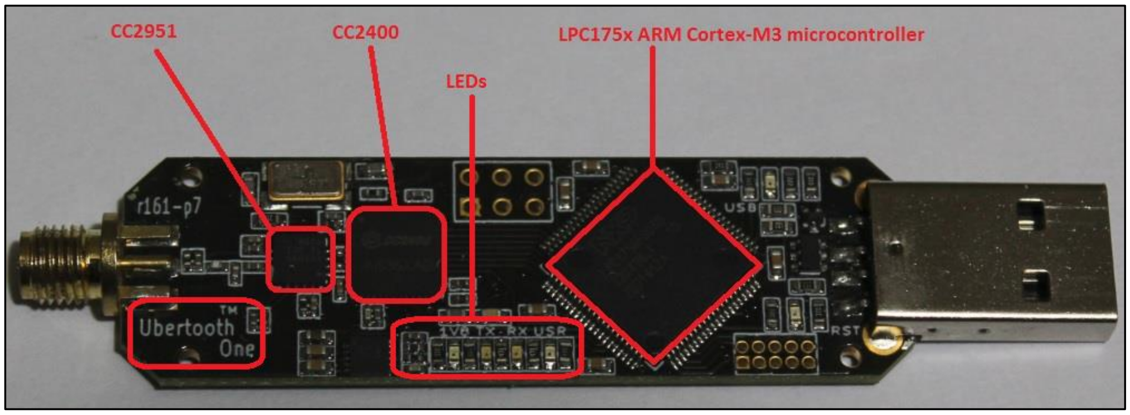


Figure 7: Ubertooth One, showing key components

The Ubertooth One is based on the Texas instruments CC2400 which demodulates raw bits from the air and stream them to a microcontroller. Unfortunately, the Ubertooth hardware is incompatible with Bluetooth Enhanced Data Rate (EDR) data modulations.

Even though this EDR limitations exists, fortunately Bluetooth Low Energy does work. “Although it was originally built to monitor classic Basic Rate (BR) Bluetooth, it serves as an excellent platform for building a BLE sniffer” (Ryan, 2014). Besides the CC2400 other key components and the device architecture are described in more detail in Table 16.

Ubertooth One Architecture (Ossmann, 2013)**:**
• RP-SMA RF connector: connects to test equipment, antenna or dummy load. • CC2591 RF front end (from Texas Instruments). • CC2400 wireless transceiver (from Texas Instruments). • LPC175x ARM Cortex-M3 microcontroller with Full-Speed USB 2.0 • USB A plug: connects to host computer running Kismet or other host code.

Table 16: Ubertooth one architecture and features (Ossmann, 2014)

The indicator LED’s deliver useful information as to whether the device is functioning correctly or not, the LED’s are described in more detail in Table 17.

Ubertooth One LED Guide (Ossmann, 2013)**:**
• RST: indicates that the LPC175x is powered on. This should always be on during operation except during a full reset of the LPC175x (e.g., while entering ISP mode). • 1V8: indicates that the CC2400 is being supplied with 1.8 V. Control of this supply depends on firmware. 1V8 power required to activate the crystal oscillator which is required to activate USB. • USB: indicates that USB has passed enumeration and configuration. • TX: Control of this LED depends on firmware. It typically indicates radio transmission. • RX: Control of this LED depends on firmware. It typically indicates radio reception. • USR: Control of this LED depends on firmware.
The TX, RX, and USR LEDs blink in a distinctive chasing pattern when the bootloader is ready to accept USB DFU commands.

Table 17: Ubertooth One LED Guide (Ossmann, 2014)

The prerequisite software for this device was already installed, as described in the Software Requirements section above. The Ubertooth was then plugged in and briefly tested to ensure it worked as expected. "Overall, we have a very effective method of determining the master's UAP through passive monitoring, It is complicated, but is only a small part of the even more complicated process of determining a piconet's frequency hopping pattern and hopping along" (Ossmann, 2014).

3.3.2 Linksys USBBT100

On the recommendation of 2600 Hackers' Quarterly magazine article, **Bluetooth Hacking Primer**, two class 1 Linksys USBBT100 USB Bluetooth adaptor were purchased - "try to get a Class 1 adapter that has an external antenna, such as the Linksys USBBT100. Adapters with external antennas are obviously going to have a better range out of the box, but are also easier to modify for use with a larger antenna" (Nardi, 2010). Most vendors do not design dongles with external antenna connectors, however, with a pigtail/antenna attached, the range of a Class 1 dongle can be extended. The intention was to attach a pigtail to one of the adaptors, enabling a larger external antenna to be mounted, "Often, you can modify a standard Bluetooth dongle to add an external antenna connector using a soldering iron and basic hardware hacking skills" (Cache, et al., 2010, p. 281). Bluetooth devices operate in the 2.4 GHz spectrum, so can use antennas designed for WLAN devices.

Technical specifications for the USBBT100 can be found in Table 18 below. It should be noted, that while both these devices were from Linksys, the unmodified device used a chipset from Cambridge Silicon Radio (CSR), while the modified device (with a pigtail) used a Broadcom Corporation chipset. Besides the use of different chipsets, the HCI version and LMP version was also different. The table offers a comparison of the main features of both devices. Information was also taken from the **hciconfig –a** command, the full output of which is displayed in Appendix H and Appendix I.

Name	Linksys Unmodified Device	Linksys Modified Device
Manufacturer	Linksys	Linksys
Name	USBBT100	USBBT100
Power Class	Class 1 (13~17dBm)	Class 1 (13~17dBm)
Antenna	1.2 dBi	5 dBi (attached to pigtail)
BD Address	00:0C:41:E2:77:7B	00:13:10:5D:3F:55
HCI Version	1.1	1.2
LMP Version	1.1	1.2
Manufacturer ID	Cambridge Silicon Radio (10)	Broadcom Corporation (15)
Bluetooth Specification	Bluetooth Core Specification 1.1	Bluetooth Core Specification 1.1

Table 18: Linksys USBBT100 Specifications

Further investigation of these HCI and LMP versions, indicated that both these device support the Bluetooth Core Specification 1.1, this means that the Data Transfer rate is limited to 1 Mbit/s. Figure 8 below shows the unmodified Linksys dongle, which used the CSR chipset.

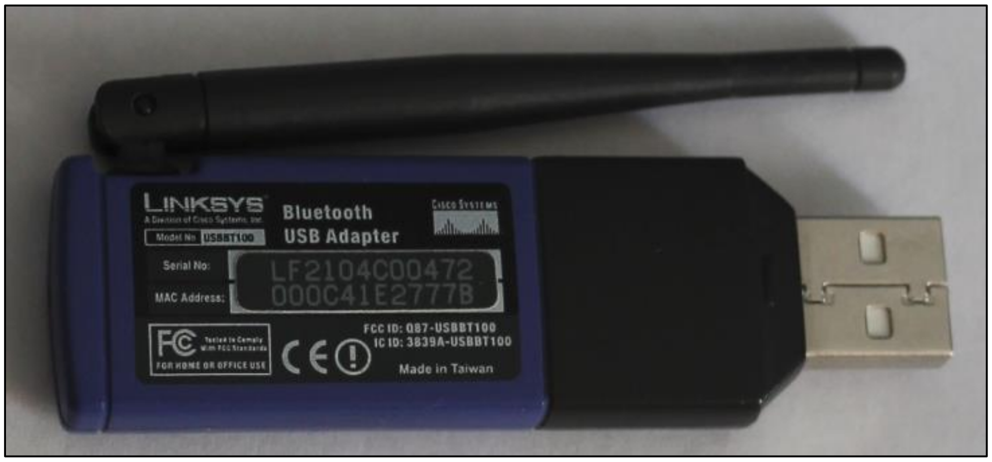


Figure 8: Linksys USBBT100 unmodified device (BD_ADDR: 00:0C:41:E2:77:7B)

Shown next in Figure 9, is the Linksys device that was eventually fitted with an external pigtail adaptor, to allow external antennas. The Broadcom chip itself is highlighted in red.

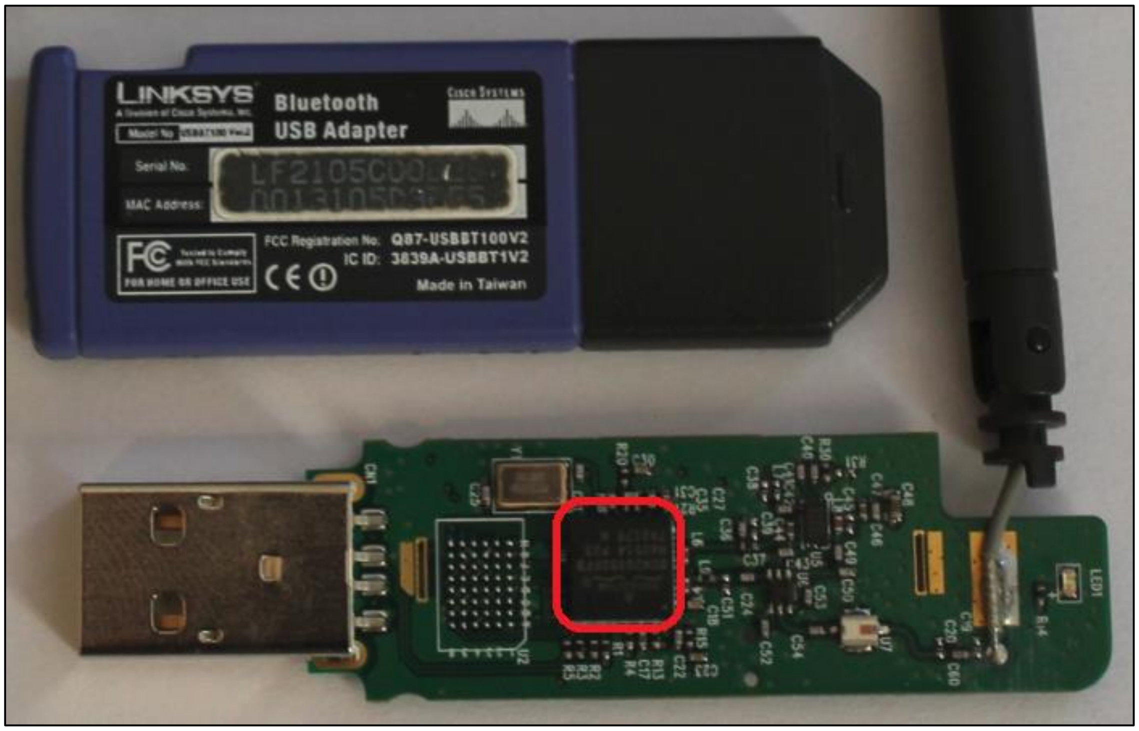


Figure 9: Linksys USBBT100 modified device (BD_ADDR: 00:13:10:5D:3F:55)

Figure 10 below shows a comparison of the two devices, the device on the left is the unmodified Linksys dongle, with its 1.2 dBi antenna, while the device on the right shows the modified Linksys dongle, with a much larger 5 dBi antenna attached to the pigtail.

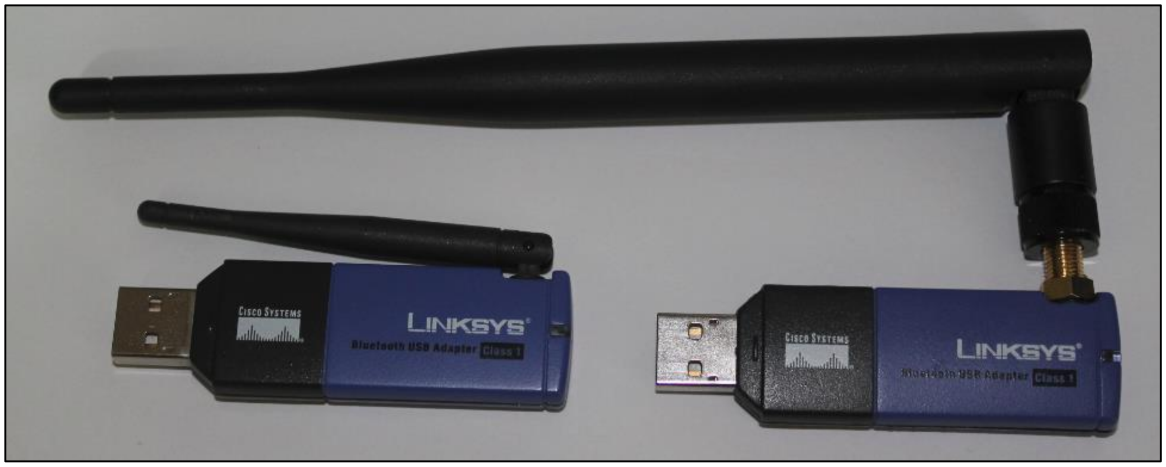


Figure 10: Linksys USBBT100 unmodified and modified devices (pigtail soldered in place)

3.3.3 Aircable Host XR

On the guidance of another 2600 Hackers' Quarterly magazine article, **The Bluetooth Hunter's Guide**, an Aircable Host XR was purchased; "My main workhorse is the Aircable Host XR, an extremely powerful USB Bluetooth device that is primarily designed for proximity marketing. It has a 200 mW radio (twice the power of a normal Class 1 device) and a standard RP-SMA antenna connector, which makes it perfect for long range applications" (Nardi, 2012). As this device had the most powerful radio transmitter it was used with the larger 9 dBi Antenna. With the device plugged in, the **hciconfig –a** command was run, output shown in Appendix J, confirming the presence of the CSR chipset. The specifications for this device are summarised in Table 19 below.

Name	Aircable Host XR
Manufacturer	Aircable
Name	Host XR
Power Class	Class 1 (19.5 dBm)
Antenna	9 dBi
BD Address	00:50:C2:7F:47:80
HCI Version	2.0
LMP Version	2.0
Manufacturer ID	Cambridge Silicon Radio (10)
Bluetooth Specification	Bluetooth Core Specification 1.2

Table 19: Aircable Host XR Specifications

The HCI and LMP versions taken from running the hciconfig command (shown in Appendix J) indicated that this device supports the Bluetooth Core Specification 1.2, this means that the Data Transfer rate is limited to 1 Mbit/s. Figure 11 below shows the printed circuit board used by the Aircable, the CSR chip and Lower Address Part are highlighted.

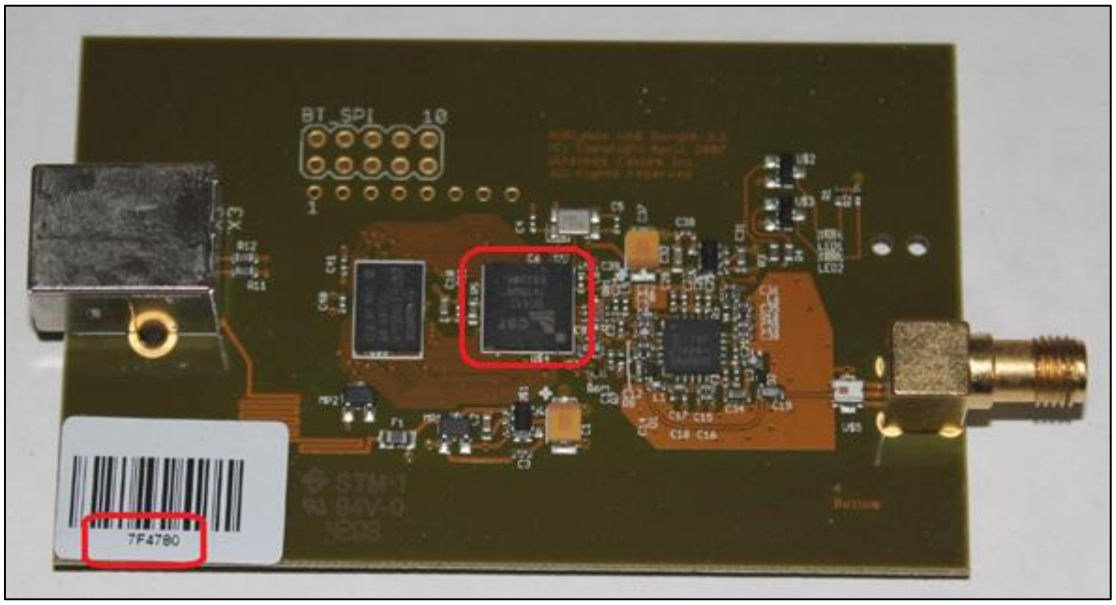


Figure 11: Host XR, showing Lower Address Part (LAP) – 7F4780 (BD_ADDR: 00:50:C2:7F:47:80)

3.3.4 SENA Parani UD-100

Besides the Aircable, according to Wright and Cache (2015) only a limited number of commercial Bluetooth adaptors are available with external antenna connectors, and are typically intended for industrial type applications. One product is the SENA Parani UD-100 adaptor with a RP-SMA antenna connector. This product also has the advantage of using the CSR chipset. Based on the recommendation included in **Hacking Wireless Exposed**, both second and third editions, a Sena was sourced.

"The Parani UD100 from SENA is a high performance Class 1 Bluetooth adapter that can extend the effective range of Bluetooth up into the hundreds of metres. This particular Class 1 adapter is much smaller and lighter than other high performance hardware from companies such as Aircable, which makes it a natural choice for mobile work" (Nardi, 2014). Included in the purchase was a stub 1 dBi antenna (not used in testing). A 5 dBi antenna was attached to the Sena, during the testing phase. This proved a discrete and powerful device, a summary of its specifications can be found in Table 20 below.

Name	SENA Parani UD-100
Manufacturer	SENA
Name	Parani UD100
Power Class	Class 1 (19dBm) + 6dBm EDR
Antenna	5 dBi
BD Address	00:01:95:21:C4:95
HCI Version	4.0
LMP Version	4.0
Manufacturer ID	Cambridge Silicon Radio (10)
Bluetooth Specification	Bluetooth Core Specification 2.1 + EDR

Table 20: SENA Parani UD100 specifications

The HCI and LMP versions taken from running the hciconfig command (shown in Appendix K) indicated that this device supports the Bluetooth Core Specification 2.1 + EDR, this means that the Data Transfer rate was limited to 3 Mbit/s. The Sena device is shown in the Figure 12 below, note that the Lower Address Part (LAP) of the BD_ADDR is also displayed on the casing.

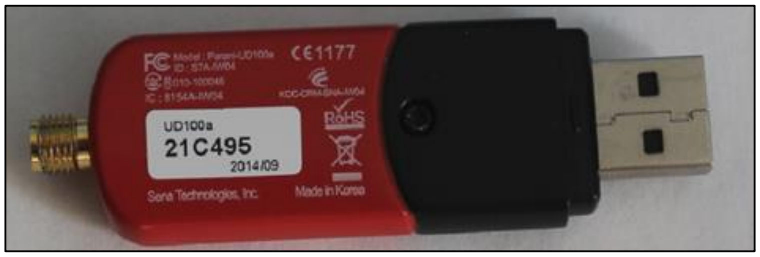


Figure 12: SENA Device, showing Lower Address Part (LAP) - 21C495 (BD_ADDR: 00:01:95:21:C4:95)

Besides displaying the LAP on the external casing, the same information was displayed on the printed circuit board, as shown in Figure 13. Also highlighted is the Cambridge Silicon Radio chip.

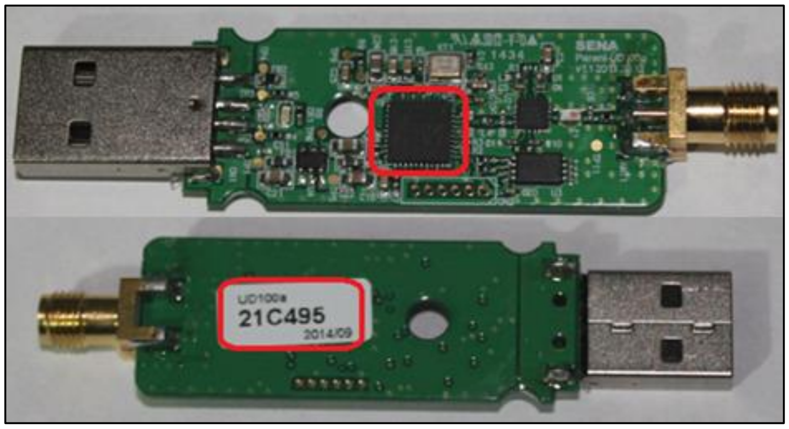


Figure 13: SENA device, again showing Lower Address Part (LAP) - 21C495

3.3.5 Antennas

Several antennas were used for this project, including those shown in Table 21. Note that the ranges shown were estimates specifically for the Sena Parani UD-100 device.

Antenna	Gain	Range (UD-100)
Stub antenna	1 dBi	~ 300 m
Omni-Directional Dipole Antenna	3 dBi	~ 400 m
Omni-Directional Dipole Antenna	5 dBi	~ 600 m
High-gain Omni-directional Antenna	9 dBi	> 1 km

Table 21: Antenna Types

As noted by Ossmann (2014), at a minimum the stub antenna, or greater, should always be used when a device is powered one, this is especially important for the Ubertooth One.

3.3.6 Lenovo Thinkpad L412

During the initial stages of this thesis, Kali Linux was successfully installed in the VirtualBox application on a Windows 7 machine, however, even though Kali installed and ran successfully, it proved extremely difficult to get this running image working successfully with the Ubertooth One attached.

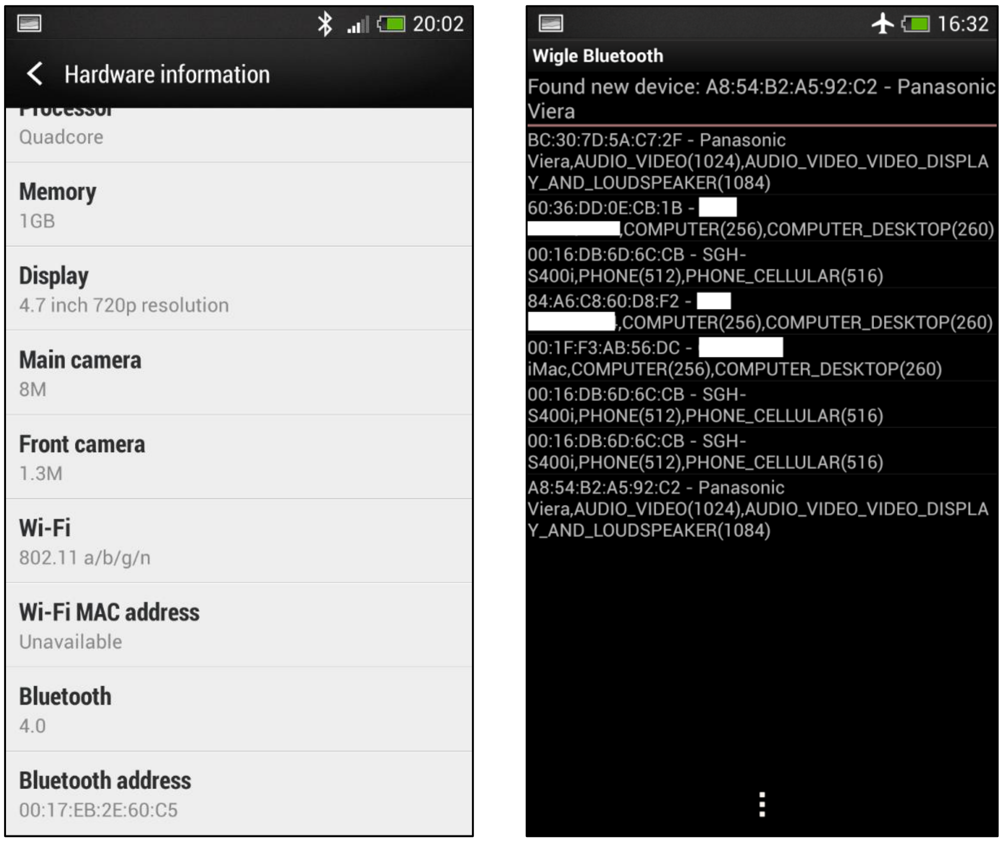
For this reason a Lenovo Thinkpad L412 laptop was sourced. Installing Kali was again a straightforward process, and once installed, this Linux Operating System immediately picked up the Ubertooth One, plus all the additional Bluetooth devices without any difficulty.

The laptop had an Intel Core i5 CPU M 520 processor running at 2.4GHz, which is a 64 bit processor that contains 2 cores and runs on 4 threads. The laptop contained 8GB RAM, and a 256 GB Crucial MX100 SSD Hard drive.

CHAPTER 4 - Design of Experiment

4.1 Introduction

“Due to the ad-hoc and decentralised nature of Bluetooth technology, administrators are often unaware of the amount of Bluetooth technology in use, and their exposure to Bluetooth attacks. While many organisations disregard Bluetooth threats, thinking the technology is limited to short-range communication, the reality is that tests have shown it is possible for an attacker to communicate to a short-range Bluetooth device from over a mile away!” (Wright, 2007). While it is possible to use a Bluetooth enabled android phone and free google playstore software, such as Wigle Bluetooth, as shown in the subsequent Figure, it’s important to note that this hardware is most likely a class 2 device, and Wigle will only pick up on devices that are in discoverable mode. To the left is the Bluetooth settings for the author’s phone, to the right is Wigle Bluetooth, a couple of PC names were redacted by the author.



20:02

Hardware information

Processor
Quadcore

Memory
1GB

Display
4.7 inch 720p resolution

Main camera
8M

Front camera
1.3M

Wi-Fi
802.11 a/b/g/n

Wi-Fi MAC address
Unavailable

Bluetooth
4.0

Bluetooth address
00:17:EB:2E:60:C5

16:32

Wigle Bluetooth

```
Found new device: A8:54:B2:A5:92:C2 - Panasonic
Viera
BC:30:7D:5A:C7:2F - Panasonic
Viera,AUDIO_VIDEO(1024),AUDIO_VIDEO_VIDEO_DISPLA
Y_AND_LOUDSPEAKER(1084)
60:36:DD:0E:CB:1B - [redacted]
[redacted],COMPUTER(256),COMPUTER_DESKTOP(260)
00:16:DB:6D:6C:CB - SGH-
S400i,PHONE(512),PHONE_CELLULAR(516)
84:A6:C8:60:D8:F2 - [redacted]
[redacted],COMPUTER(256),COMPUTER_DESKTOP(260)
00:1F:F3:AB:56:DC - [redacted]
iMac,COMPUTER(256),COMPUTER_DESKTOP(260)
00:16:DB:6D:6C:CB - SGH-
S400i,PHONE(512),PHONE_CELLULAR(516)
00:16:DB:6D:6C:CB - SGH-
S400i,PHONE(512),PHONE_CELLULAR(516)
A8:54:B2:A5:92:C2 - Panasonic
Viera,AUDIO_VIDEO(1024),AUDIO_VIDEO_VIDEO_DISPLA
Y_AND_LOUDSPEAKER(1084)
```

Figure 14: Android Bluetooth settings and Wigle Bluetooth in action

Figure 14 above demonstrates that even low powered equipment, and active scanning, can produce results, however, the author found the Wigle software quite flaky, and prone to unexpected crashes.

This experiment used some off-the-shelf hardware, the Ubertooth One, in combination with some simple Ubertooth and BlueZ Linux commands to examine potential exposure risks via Bluetooth. The author aims to affirm Nardi's assertion (2012) that the end result is that there are still many Bluetooth devices announcing their presences to anyone who listens.

4.2 Testing Setup

In order to facilitate this testing phase, it was decided to plug in all the devices simultaneously, and simply bring them all down initially, using the **hciconfig hciX down** command, as detailed in the above section 3.2.2. To facilitate this, it was necessary to use a powered hub, as shown in Figure 15 below.

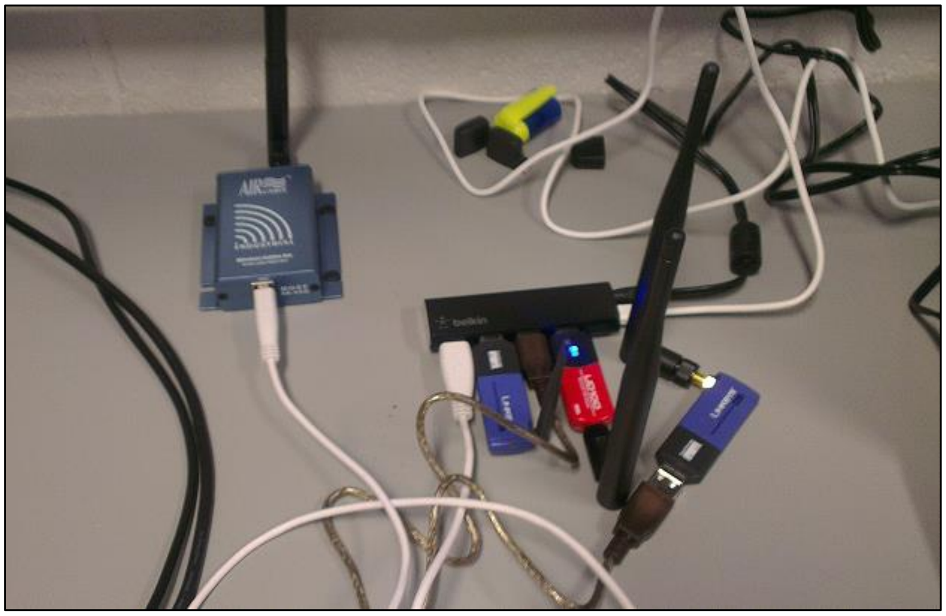


Figure 15: Equipment in use at Letterkenny Institute of Technology.

A powered hub was important for this, as the laptop only had 3 USB ports in total, and the Ubertooth was using one of those. Note that that the Ubertooth is not shown in the Figure above, as it was plugged directly into the laptop. An alternative setup may involve using a docking station, as shown in Figure 16 below. Note the yellow-cased Ubertooth one on the left, plugged directly into the laptop, while the other four devices are plugged directly into the Thinkpad L412 docking station.

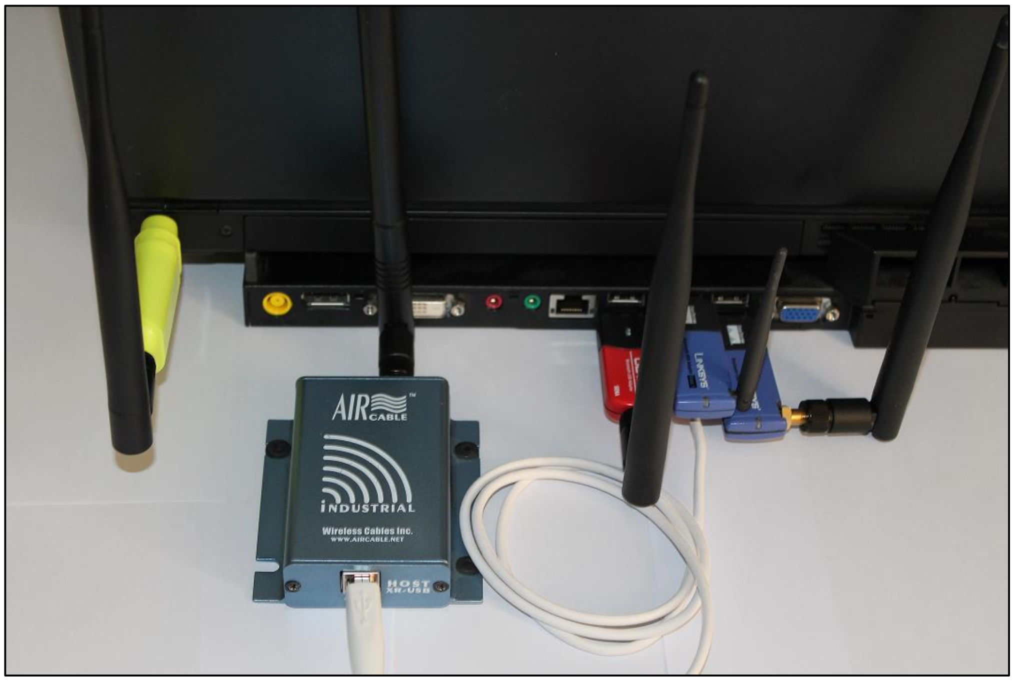


Figure 16: USB devices plugged into docking station

Each device could then be brought back up individually, using **hciconfig hciX up**, and tested in combination with the Ubertooth One. Performing the testing in this way allowed for all the commands to be run from a *Bourne Again Shell* (Bash) shell script. This would allow the author to run the script, taking several hours, without the need for the author to be physically present. It is very important to record the hciconfig configuration data at the start of each run, as this can vary each time the machine is rebooted, and may be different each time. To help with this task each device was individually identified first, the identification information captured is recorded in Table 22 below.

Device Name	Bluetooth Device Address	Vendor
Device 1: Thinkpad Bluetooth Device	BD_ADDR: **78:DD:08:B2:DE:4C**	Hon Hai Precision Ind. Co.,Ltd.
Device 2: Linksys USBBT100	BD_ADDR: **00:0C:41:E2:77:7B**	Cisco-Linksys, LLC (was: The Linksys Group, Inc.)
Device 3: Linksys USBBT100 (modified)	BD_ADDR: **00:13:10:5D:3F:55**	Cisco-Link
Device 4: Parani UD100	BD_ADDR: **00:01:95:21:C4:95**	Sena Technologies
Device 5: Aircable Host XR	BD_ADDR: **00:50:C2:7F:47:80**	ieee registration authority

Table 22: Bluetooth test devices individual BD_ADDR Addresses

It should be noted, that the vendor column in the above table was discovered using the first 6 characters of the BD_ADDR (NAP and UAP), and submitting a query at the **mac find** (Coffer, 2013) website. Besides using the MAC address to discover the manufacturer of the device, it's also possible to discover the chipset maker using the official Bluetooth company identifiers, as described in the **company identifiers** webpage (Bluetooth SIG, 2015), and summarised in Table 23.

Company ID		
Decimal	**Hexadecimal**	**Company**
10	0x000A	Cambridge Silicon Radio
15	0x000F	Broadcom Corporation

Table 23: Company Identifiers

4.3 Test Run

The venue chosen was *Letterkenny Institute of Technology* (LYIT), as this would not be seen to impact on the security of a private company or business. It was expected that the close proximity traffic would have Bluetooth enabled laptops/tablets/mobile-phones, due to the student population at the Institute. The location where the test equipment was run was **Room 2277**, which was in close proximity to several corridors with fairly frequent passing traffic. This location is highlighted in Figure 17.

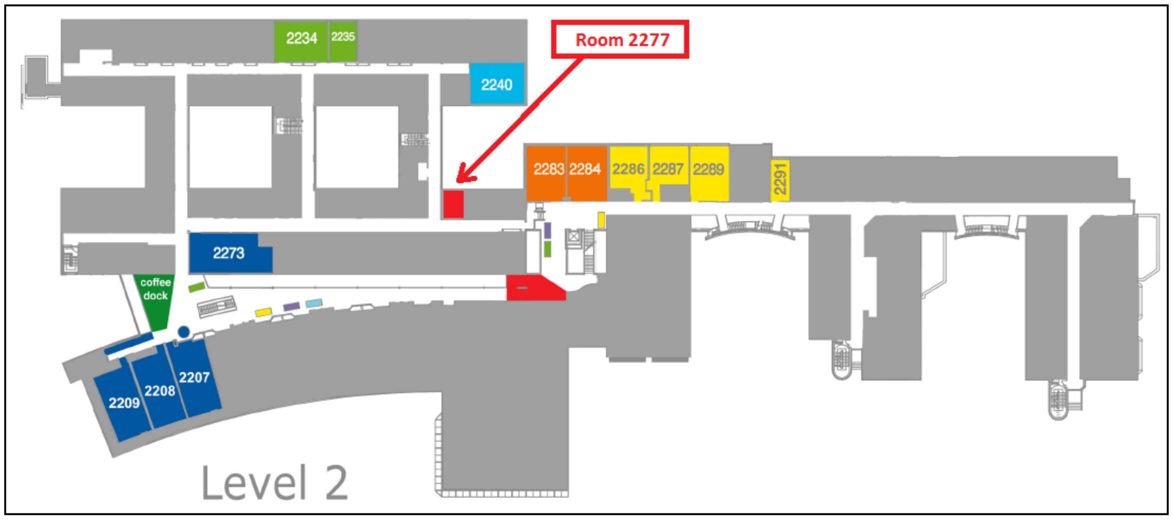


Figure 17: Location of test equipment at LYIT (LYIT, 2015)

Several bash scripts were created. For the formation of these scripts it's important to note the following piece of information: "ubertooth-scan: This allows you to identify devices in hidden-mode/non-discoverable mode. You need an additional *hciX* interface, as the Ubertooth is not a fully-fledged BT dongle - just a sniffer; Here the Ubertooth grabs LAP & UAP to form addresses, and hands off inquiry to a proper BT dongle" (Davies, 2013). To handle the individual Bluetooth dongles, the **hciconfig** command was used for bringing up/down of the attached devices, allowing each their turn to work with the Ubertooth One.

ubertooth-scan was run with several command line options, the **–b** option was used to select which attached device to use, while the **–t** option set a time limit on the duration of the scan (900 for example means 900 seconds, or 15 minutes). The **–s** option was used to perform a HCI type scan. The **–x** option turned on the extended query functionality. "I've also added an extended query (triggered by the -x option) which will check the device for supported features, chipset version and clock offset from the local device. Using a dongle to get the clock offset for a remote device allows us to calculate the clock value of the target and use that to hop along with the piconet, dumping packet data to screen as we go." (Spill, 2012).

The testing was run over three phases, with each phase attempting to improve the scripts used. The scripts used during each phase, and their respective component time are described in Table 24.

Test Phase	Appendix	bash scripts	Component Commands	Time

One	Appendix L	scan1.bash	`ubertooth-scan -b $j -t 1800`
			`ubertooth-scan -b $j -t 1800 -s`
			`ubertooth-scan -b $j -t 1800 -x`
				1800s (30 mins)

Two	Appendix M	scan2.bash	`ubertooth-scan -b $j -t 900`
			`ubertooth-scan -b $j -t 900 -s`
			`ubertooth-scan -b $j -t 900 -x`
				900s (15 mins)

Three	Appendix N	scan_master.bash		
		scan_slave1.bash	`ubertooth-scan -b $1 -t 900`	900s (15 mins)
		scan_slave2.bash	`ubertooth-scan -b $1 -t 900 -s`	900s (15 mins)
		scan_slave3.bash	`ubertooth-scan -b $1 -t 900 -x`	900s (15 mins)

Table 24: Testing scripts and their component times

The last step before running each command was to change the scripts permission to executable, using the Linux **chmod** command. Each script could then be run on the command line, for example: `root@kali:~# ./scan_master.bash`

4.4 Summary

With the equipment correctly configured and ready to go, initial testing was performed. A number of echo statements were included within each script, for the purpose of logging and confirming the configuration at runtime. This logging is very important, as the hciX number assigned to each device can change each time the machine is rebooted or powered off, although it is possible to avoid this by updating the `/etc/bluetooth/hcid.conf` configuration file (Vanrenterghem, 2015).

Once a quick check of the file was run on the command line, it could be killed with a CTRL-C command, and then run for real, with the output piped to a log file for later analysis. For example:

```
root@kali:~# ./scan_master.bash > output_Run3c.txt
```

By combining relatively inexpensive off-the-shelf hardware, with some basic Bluetooth commands, within some shell scripts, the combination of these three elements should prove a powerful combination in determining whether Bluetooth devices are in the locality, whether discoverable or non-discoverable. By logging this information, a hardware inventory can be created and updated, which can help determine/improve the compliance stature of an organisation.

CHAPTER 5 - Results and Discussion

5.1 Introduction

In order to interpret the raw data from the test runs, it needed to be processed. A perl script was used for this purpose, and the latest version of Activeperl was downloaded from the Activestate website. Perl was a good choice for this as it inherits features from sed and awk, as well as having a powerful regular expression engine. The script that was created, called **bluetooth.pl** is displayed in Appendix O. The general purpose of this script was simply to parse the text output from the above scans, and output the relevant elements into a comma-separated file, which could then be delivered for further processing using Microsoft Excel features. The script could be run on the command line as follows:

```
perl bluetooth.pl EXCEL "sena1.txt" >sena1.csv
```

(Note: that by swapping the EXCEL option for SYSLOG on the command line, a SYSLOG type output could be produced). A sample of how the script runs is shown in Table 25.

Sample Input Data

systime=1426002783 ch=54 LAP=a192c1 err=2 clk100ns=2328861729 clk1=12955530 s=-86 n=-88 snr=2
systime=1426002793 ch=13 LAP=808afb err=2 clk100ns=2427478348 clk1=12971308 s=-79 n=-85 snr=6
systime=1426002919 ch=59 LAP=a03035 err=2 clk100ns=411314042 clk1=13173010 s=-86 n=-89 snr=3
systime=1426002961 ch=63 LAP=764f8c err=2 clk100ns=830427841 clk1=13240068 s=-88 n=-88 snr=0

Sample Output Data
1426002783,'a192c1 1426002793,'808afb 1426002919,'a03035 1426002961,'764f8c

Table 25: bluetooth.pl functionality

Creating this script was not without its problems though, it can be noted from the preceding table that there was a leading single quote ‘ before the LAP entry. This was deliberately included, due to the problems Microsoft Excel had in interpreting different cell entries. For example, several cells were auto-formatted as date cells, or with exponential type numbers in them, or with the leading zeros removed, if the leading single-quote symbol was not used, these errors would have had to be repaired manually.

Another issue encountered, was a small error in the regular expression used in the script. The offending sections of the regular expression is shown in Table 26, highlighted in red.

Original Regular Expression
`($line=~m/^systime=(\d+)\sch=(\d+)\sLAP=(\w+)\serr=(\d+)\sclk100ns=(\d+)\sclk1=(\d+)\ss=(-\d+)\sn=(-\d+)\ssnr=(\d+)/)`
Repaired Regular Expression
`($line=~m/^systime=(\d+)\sch=\s?(\d+)\sLAP=(\w+)\serr=(\d+)\sclk100ns=(\d+)\sclk1=(\d+)\ss=(-\d+)\sn=(-\d+)\ssnr=(\d+)/)`

Table 26: Broken and repaired regular expression

The addition of the `\s?` element proved vital, as it helped avoid multiple entries from getting inadvertently dropped by the perl script. In Table 27 below, for example, the two lines highlighted in red would have been dropped, simply because there was a space between ch= and its associated channel number, which was not catered for by the originally coded regular expression.

Dropped data example
systime=1426003356 ch=33 LAP=22aa1c err=2 clk100ns=1508533286 clk1=13872853 s=-65 n=-86 snr=21
systime=1426003356 ch=78 LAP=22aa1c err=2 clk100ns=1513080480 clk1=13873581 s=-75 n=-86 snr=11
systime=1426003356 ch= 1 LAP=22aa1c err=2 clk100ns=1513282813 clk1=13873613 s=-63 n=-83 snr=20
systime=1426003357 ch= 5 LAP=22aa1c err=1 clk100ns=1513659567 clk1=13873673 s=-63 n=-83 snr=20
systime=1426003357 ch=13 LAP=22aa1c err=2 clk100ns=1514445198 clk1=13873799 s=-64 n=-84 snr=20
systime=1426003357 ch=19 LAP=22aa1c err=0 clk100ns=1515056474 clk1=13873897 s=-66 n=-86 snr=20

Table 27: Dropped data example

Once the perl issues had been ironed out, in order to run the script in a more automated fashion, and to help reduce error, a windows batch file was created to work through the data. The main purpose of the batch was to run the perl script iteratively to create a number of *Comma Separated Value* (CSV) files. This batch file could then be used for data processing for progressive runs, and help avoid errors in typing. A sample of this batch file, called **run.bat**, is included in Appendix P.

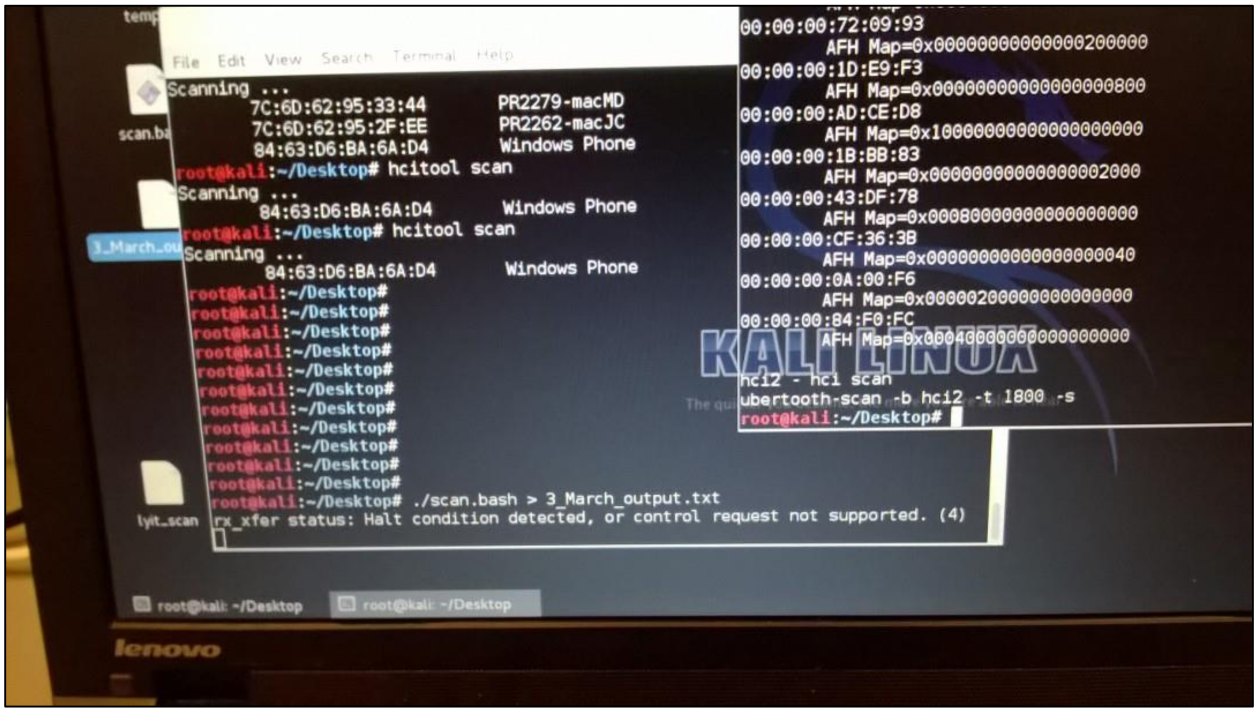
5.2 Testing Phase 1

Testing Phase 1 focused on the results of running **scan1.bash**, shown in Appendix L. As already noted, it was important to keep a record of the configuration during each run, as this could change between machine reboots. For the first phase of testing, the devices were picked up as shown in Table 28.

Host Controller Interface	Bluetooth Device Address	Physical Device
hci0	78:DD:08:B2:DE:4C	Thinkpad Bluetooth Device
hci1	00:01:95:21:C4:95	SENA Parani UD100
hci2	00:13:10:5D:3F:55	Linksys USBBT100 (modified)
hci3	00:0C:41:E2:77:7B	Linksys USBBT100
hci4	00:50:C2:7F:47:80	Aircable Host XR

Table 28: Testing Phase 1 device configuration

The first run was for investigative purposes mainly. After some teething problems, the scripts were eventually run. Unfortunately, after several hours the script halted abruptly, as shown in Figure 18.

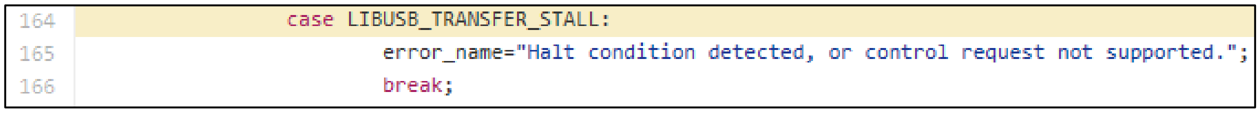


```
Scanning ...
        7C:6D:62:95:33:44       PR2279-macMD
        7C:6D:62:95:2F:EE       PR2262-macJC
        84:63:D6:BA:6A:D4       Windows Phone
root@kali:~/Desktop# hcitool scan
Scanning ...
        84:63:D6:BA:6A:D4       Windows Phone
root@kali:~/Desktop# hcitool scan
Scanning ...
        84:63:D6:BA:6A:D4       Windows Phone
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop#
root@kali:~/Desktop# ./scan.bash > 3_March_output.txt
rx_xfer status: Halt condition detected, or control request not supported. (4)
```

```
00:00:00:72:09:93
            AFH Map=0x00000000000000000200000
00:00:00:1D:E9:F3
            AFH Map=0x00000000000000000000800
00:00:00:AD:CE:D8
            AFH Map=0x10000000000000000000000
00:00:00:1B:BB:83
            AFH Map=0x00000000000000000002000
00:00:00:43:DF:78
            AFH Map=0x00080000000000000000000
00:00:00:CF:36:3B
            AFH Map=0x00000000000000000000040
00:00:00:0A:00:F6
            AFH Map=0x00000200000000000000000
00:00:00:84:F0:FC
            AFH Map=0x00040000000000000000000
hci2 - hci scan
ubertooth-scan -b hci2 -t 1800 -s
root@kali:~/Desktop#
```

Figure 18: Test run error message

A quick google search for the error message **rx_xfer status: Halt condition detected, or control request not supported**, pointed the author to the Ubertooth page on github (Ossmann, 2015) that contained the code for **ubertooth.c**, and the specific code snippet is captured in Figure 19.



```
		case LIBUSB_TRANSFER_STALL:
			error_name="Halt condition detected, or control request not supported.";
			break;
```

Figure 19: Halt condition error code

The code and error message appears to indicate a problem with USB communicating with the Ubertooth One. A reboot was all that was required to fix the problem. It should be noted that the Ubertooth code is still under active development, so occasional problems could be encountered. It was also noted, that the duration coded for these runs, **-t 1800**, (30 minutes) was too long for testing purposes, particularly if any problems were encountered. This time will be reduced in later testing phases.

Once the CSV data had been loaded into the Microsoft Excel spreadsheet it needed to be labelled correctly, labelling was based off two things – the attached device name, and the type of scan being performed. It was also important at this stage to use Excel’s **Remove Duplicates** functionality (based on the LAP column only), to ensure what remained were unique LAP entries.

5.2.1 Phase 1 Results

Due to the errors encountered, and the long run times, only two devices completed the three different scan types – the internal laptop device, and the Sena device, but the modified Linksys device did manage to get through a complete Ubertooth only scan. Due to the incomplete device run through, phase 1 was considered as proof of concept testing for the subsequent phases.

5.2.2 Phase 1 Findings

Despite being an incomplete run, it was interesting how many non-discoverable (HCI Scan data) devices the Ubertooth was able to passively pick up, when compared to the equivalent discoverable information. This is clearly highlighted in Figure 20, which compares discoverable versus non-discoverable results for both the internal laptop device and the Sena device.

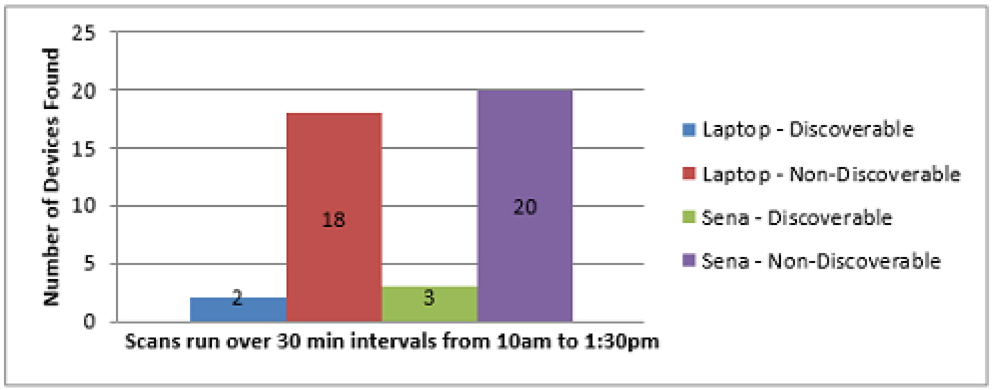


Figure 20: Discoverable versus Non-Discoverable Devices Found

5.3 Phase 2

Phase 2 concentrated on the results of running **scan2.bash**, shown in Appendix M. Again, the duplicate LAP entries were deleted and columns correctly labelled in Excel. It should be noted that this test was performed on a different date to phase one, and the time was reduced to **-t 900** (15 minutes) to ensure more time for the various planned scans. The device configuration was captured as shown in Table 29.

Host Controller Interface	Bluetooth Device Address	Physical Device
hci0	78:DD:08:B2:DE:4C	Thinkpad Bluetooth Device
hci1	00:0C:41:E2:77:7B	Linksys USBBT100
hci2	00:50:C2:7F:47:80	Aircable Host XR
hci3	00:01:95:21:C4:95	SENA Parani UD100
hci4	00:13:10:5D:3F:55	Linksys USBBT100 (modified)

Table 29: Testing phase 2 device configuration

5.3.1 Phase 2 Results

A full scan was implemented. The results of this second scan were compiled and are shown in Figure 21 below.

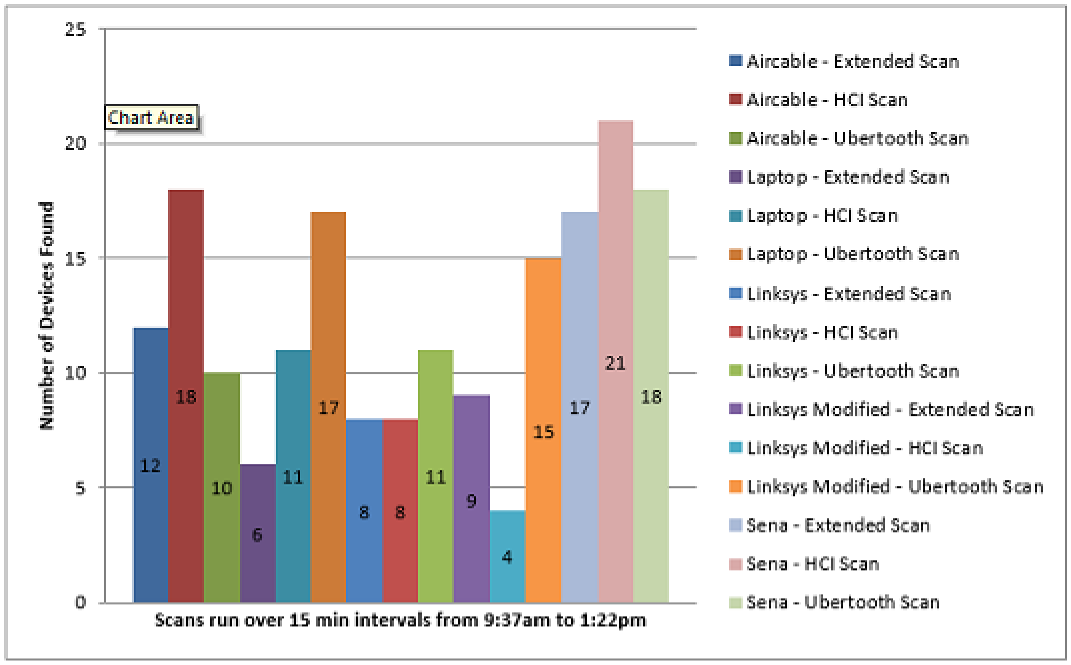


Figure 21: Phase 2 test results.

5.3.2 Phase 2 Findings

The Aircable performed well on the HCI scan, but when compared to the Sena over the three different types of scan, did not perform as well as anticipated for such a powerful device. The laptop device performed well for the Ubertooth scan, but was weak elsewhere. Apart from the Linksys modified Ubertooth scan, the two Linksys devices performed poorly, the clear winner during this phase of testing was the Sena Parani UD100.

Discoverable versus non-discoverable was again compiled, and shown in the Figure 22. The proportion of devices with Bluetooth enabled, but discovery mode turned off was again much higher than those devices with Bluetooth and discovery mode both enabled.

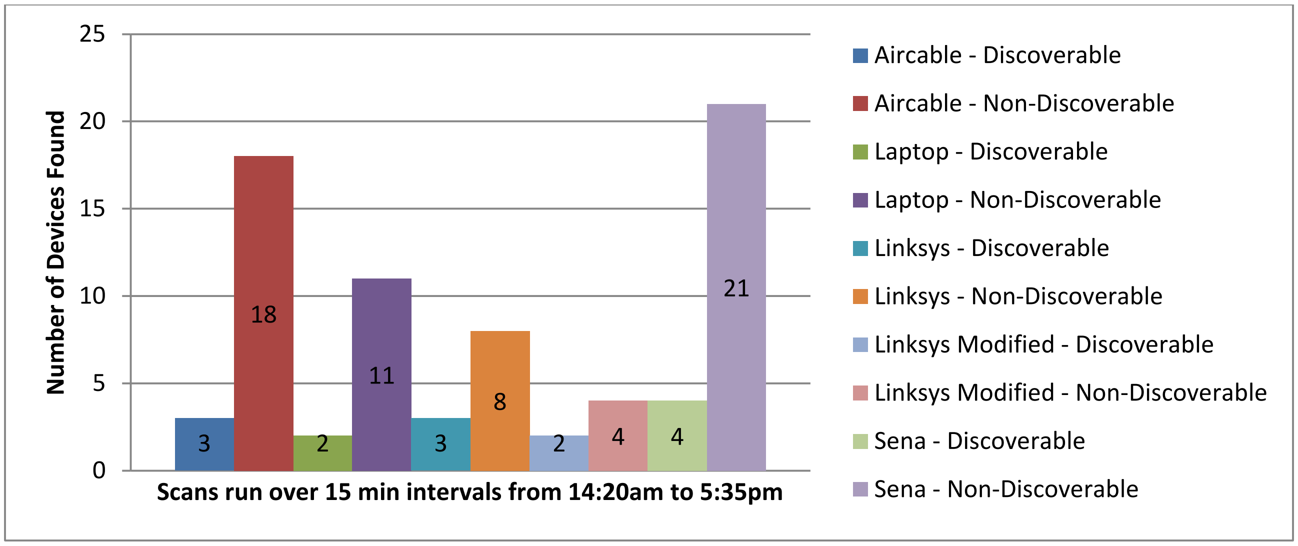


Figure 22: Discoverable versus Non-Discoverable Devices Found

5.4 Phase 3

Phase 3 concentrated on the running of the four scripts **scan_master.bash, scan_slave1.bash, scan_slave2.bash, scan_slave3.bash**, which are all displayed in Appendix N. For this phase of testing, the device configurations were recorded in Table 30 below.

Host Controller Interface	Bluetooth Device Address	Physical Device
hci0	78:DD:08:B2:DE:4C	Thinkpad Bluetooth Device
hci1	00:0C:41:E2:77:7B	Linksys USBBT100
hci2	00:50:C2:7F:47:80	Aircable Host XR
hci3	00:01:95:21:C4:95	SENA Parani UD100
hci4	00:13:10:5D:3F:55	Linksys USBBT100 (modified)

Table 30: Testing phase 3 device configuration

5.4.1 Phase 3 Results

This third run took place on the same date as phase two but at a later time. Unfortunately, due to time limitations a full scan of four devices, and a partial of the fifth device was performed for this third stage. The results of this third scan are shown in Figure 23.

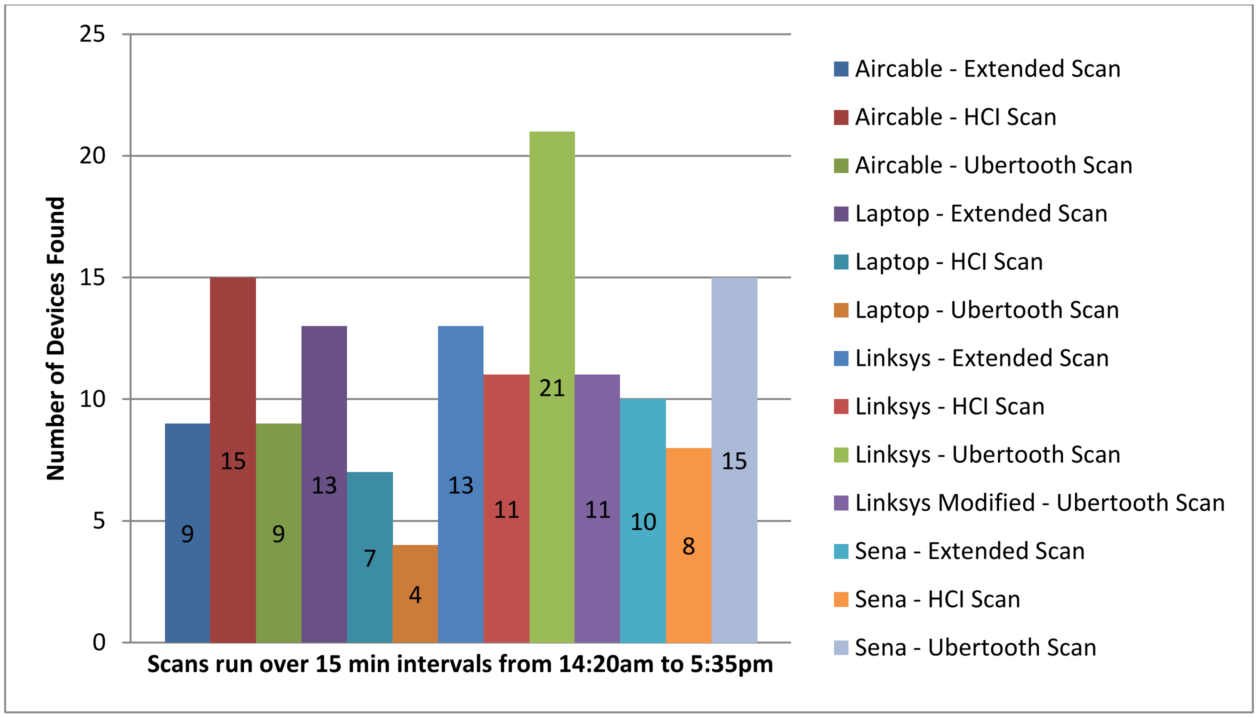


Figure 23: Phase 3 test results.

5.4.2 Phase 3 Findings

The Linksys performed well for the Ubertooth scan, there may have been a lot of proximity traffic during its run time. As expected the internal laptop device performed poorly, while the Aircable and Sena devices performed adequately, based on numbers of devices found. Again, a far higher number of non-discoverable devices were found over discoverable devices. This is displayed in Figure 24

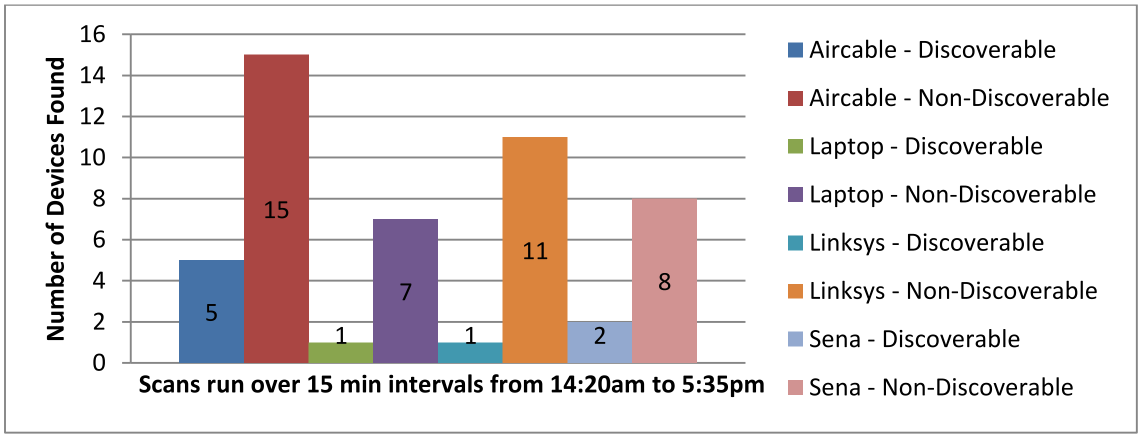


Figure 24: Discoverable versus Non-Discoverable Devices Found

5.5 Overall Results

The results of phases 2 and 3 were combined to increase the data pool, and attempt to gain accuracy. The Linksys modified device was dropped, as it did not get a full run in phase 3. This combination summed results from two separate time intervals, albeit run sequentially in both instances. Running the scans concurrently may have provided the best results, but was not possible due to limited hardware availability. Alternatively, running scans, at different time intervals, over several days, prior to summing the data may have also improved accuracy. Nevertheless, the sum totals for non-discoverable devices found is displayed in Figure 25, the Sena looks to have captured the most targets, while the internal laptop device achieved the worst results. This was in line with expectations, as class 1 devices were expected to perform better than the internal laptop device, highlighting the effective range of each device, the more powerful devices resolving a higher number of targets.

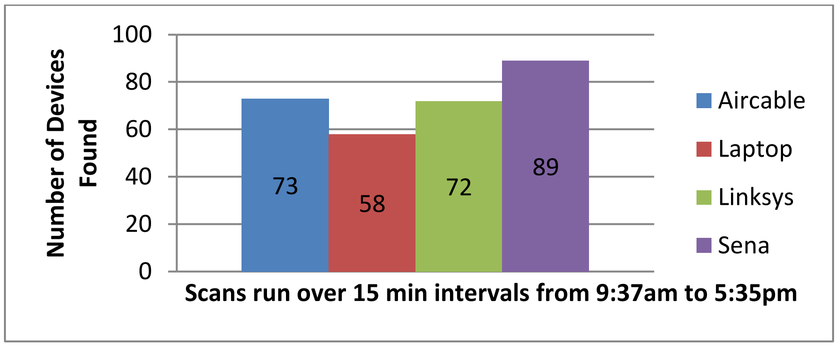


Figure 25: Combined Non-Discoverable devices found per device

The results of run 2 and run 3 were merged for discoverability too, and are displayed in Figure 26. Both non-discoverable and discoverable devices were found during the hcitool type scans only, so this was the criteria used to filter the chart. Clearly shown is a higher number of non-discoverable devices found, particularly by the Aircable and the Sena devices. The Linksys device, despite being recommended in several articles, previously cited, delivered disappointing results.

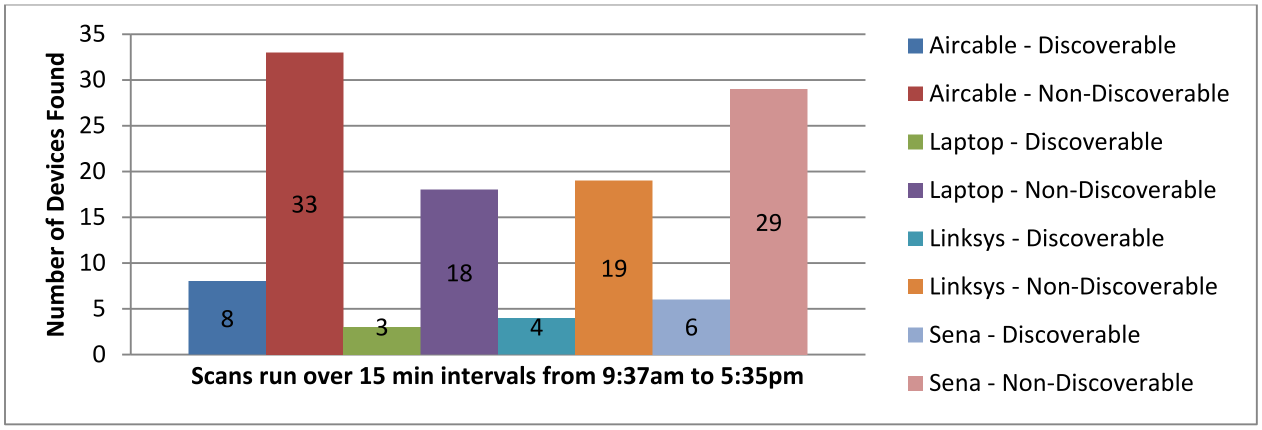


Figure 26: Combined Discoverable versus Non-Discoverable per Device

Lastly, the results in the above chart were merged for all devices, into Figure 27 below, to give a rough idea of the proportions between what could be found using a regular Bluetooth dongle (discoverable), and using an Ubertooth/Bluetooth-dongle combination (non-discoverable).

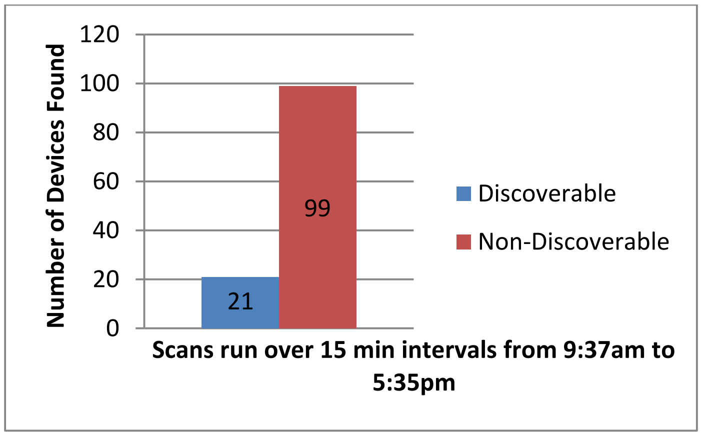


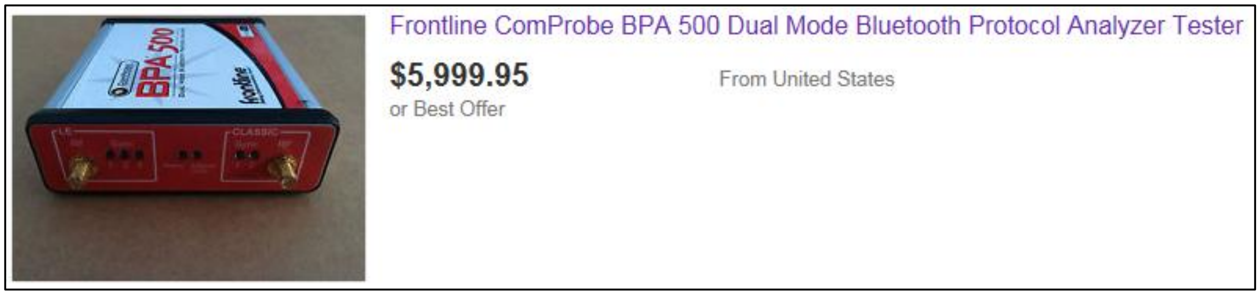
Figure 27: Combined Discoverable versus Non-Discoverable

The graph indicates that, of the Bluetooth-enable devices captured during the testing phase, 17.5% of devices were in discoverable mode, while 82.5% of devices were in non-discoverable mode. While there are several examples of Bluetooth surveys, vulnerable devices - Bluebag (Carettoni & Merloni, 2007), or discovering open Bluetooth services (Talal & Rachid, 2013) for example, they have focussed on the discoverable landscape only. Evidently, auditing Bluetooth devices with a garden variety Bluetooth device alone ignores a significant amount of potentially identifiable devices. While it won't fully complete the picture, the addition of an Ubertooth One, and it is functionality, can significantly increase the scope of any Bluetooth assessment.

CHAPTER 6 - Conclusions and Further Research

6.1 Introduction

In 2011, Joshua Wright outlined the Bluetooth Dilemma, highlighting that there is no solution for accurately detecting and enumerating non-discoverable Bluetooth devices outside of extremely expensive commercial devices. For example the Frontline ComProbe BPA 500, as shown in Figure 28. Even this device has since been superseded by the newer (and more expensive BPA 600).



Frontline ComProbe BPA 500 Dual Mode Bluetooth Protocol Analyzer Tester

$5,999.95 From United States

or Best Offer

Figure 28: Frontline ComProbe BPA 500 (2015 price)

This provides problems for auditors, “Visa can suggest merchants perform scanning, but there simply isn’t a currently available viable solution that adequately addresses the problem, not in mobile form and definitely not in a distributed enterprise-ready form” (Wright, 2011). This has now changed with the introduction of the Ubertooth One, and particularly with it working in combination with the correct Bluetooth Classic device.

6.2 Conclusion 1

Unlike WiFi, and until the production of the Ubertooth One, passive Bluetooth sniffers were unavailable to those wishing to audit from a wireless perspective (unless one was willing to pay a very large amount of money for a commercial product). Now that the Ubertooth is available, the evidence shows clearly that, in this experiment focusing on Bluetooth Classic devices alone, there are 4.7 times the amount of Bluetooth devices out there that have Bluetooth enabled but have discovery mode turned off, than those devices that have it turned on. That number underestimates the volume of Bluetooth devices in the wild, as this experiment did not attempt to capture data for EDR devices nor Bluetooth Low Energy devices.

This result shows a need for audit features beyond what a regular Bluetooth dongle alone can provide. The Ubertooth One, in combination with the correct Bluetooth hardware is capable of providing the required function of enumerating UAPs and LAPs for potential rogue devices.

6.3 Conclusion 2

The Ubertooth One is a powerful device, similar to a Class 1 Bluetooth device. Careful consideration went into the selection of test devices used. Of these devices tested in combination with the Ubertooth One, the Aircable and Sena devices appeared to be the most successful, taking into account their success with capturing data for both discoverable and non-discoverable targets.

The Linksys devices were disappointing both in terms of their success at enumerating, but also in terms of the different chipsets used in both devices despite being from the same manufacturer (Broadcom in one case and CSR in the other). An external antenna pigtail could be mounted, but detailed soldering was required. The laptop device was weak, but provided a good baseline. Both the Aircable and the Sena devices used the more favoured CSR chipset, plus they had pre-existing RP-SMA connectors. Both would be good choices for a security auditor, but given that the Sena device is cheaper than the Aircable device ($30 and $129 respectively, priced from amazon in May 2015), and given the Sena's discrete design, it is the one recommended by the author.

6.4 Strengths of the Study

This study showcased how a relatively inexpensive, off-the-shelf commodity device can enumerate multiple Bluetooth devices, even when they are in non-discoverable mode (security advice offered almost universally in the literature). "As an inexpensive device for Bluetooth analysis, the Ubertooth is a tremendously valuable tool for security analysts and attackers alike. However, it is also limited in its capabilities to capture Bluetooth Classic network activity" (Wright & Cache, 2015, p. 253). Even though this shows up potential weaknesses in the security landscape, it can also be used to great advantage in the use of logging and auditing of devices, with a view to eventually automating the output and sending SYSLOG type files to a *Security Information and Event Management* (SIEM) type system for further analysis and investigation.

6.5 Limitations of the Study

While the results outlined were valuable, it should be noted that there is a specific set of devices, the Extended Data Rate (EDR) type devices (Bluetooth v2.0 and above), which would not be picked up due to limitations of the Ubertooth hardware, specifically the Texas Instruments Chipcon CC2400 chip used as the radio transceiver interface and the limited to the demodulation capabilities of this chip. The Ubertooth can only capture Bluetooth Basic Rate traffic, it is not able to capture EDR.

The ubertooth-scan extended type scan never did return the extra information that the author expected it to, and appeared to run as a regular scan. Even though it's only supposed to run when four bytes of the address were captured (UAP and LAP), there were several instances where UAP was indeed discovered, yet additional device information was not captured. The author believes this may be due to an error in the source code.

The scans could only be run sequentially. Although it would incur significant hardware costs, more accurate results could have been achieved through the use of additional hardware, and running the scans concurrently.

Lastly, in hindsight, even though the location chosen appeared to have a good volume of passing traffic, it may have been preferable to run the experiment from a location where there was less transient traffic, and where targets were stationary (for example the College canteen).

6.6 Potential Issues for Future Research

There are several avenues where this research could be further taken, which are described in more detail below.

1. Recently, the ubertooth-scan command appears to have been upgraded to contain a new **–u** option. This allows multiple Ubertooth One devices to be connected at the same time. It would be very interesting to explore the benefits of using several Ubertooth One devices, perhaps simultaneously, and to explore the usefulness of this feature more fully. Even though these devices are not cheap, they are still far less expensive than commercial Bluetooth sniffing devices. (See next item also for similar research potential).

2. Bluetooth low energy – There is an additional command that can be used with the Ubertooth One, **ubertooth-ble**. While this thesis did focus on Bluetooth Classic, a valid avenue would be to explore the usefulness of the Ubertooth One with regards to Bluetooth Low Energy. If possible, and using two Ubertooth One devices, it would be interesting to run an ubertooth-scan and an ubertooth-ble scan simultaneously, in order to capture the packets/traffic using both technologies.
3. The ubertooth-scan extended scan never did return the extra information that the author expected it to, and appeared to simply run as a regular scan. Even though it's only supposed to run when four bytes of the address were captured (UAP and LAP), there were several instances in the results captured where UAP was discovered, yet additional device information was not captured. The author believes this may be due to an error in the source code. Further investigation could yield and answer to this problem.
4. There are several commercial devices available that can perform similar scans to the Ubertooth One, and two in particular are described in more detail in Hacking Exposed Wireless (3rd Edition), the Frontline BPA 600 Sniffer (Wright & Cache, 2015, p. 255), and the Ellisys Bluetooth Explorer 400 (Wright & Cache, 2015, p. 259). However, these devices are very expensive. It would be interesting to determine if the extra functionality they provide is worth the additional purchase price. In particular, a comparison of these devices against several Ubertooth Ones would prove interesting.
5. Tracking/Surveillance - hcitool can displaying the *Received Signal Strength Indication* (RSSI) for a given BD_ADDR, which could be used as a crude form of proximity detection. Unfortunately, due to the different output ratings of various devices you can't directly equate RSSI to a set distance. While the hcitool only works with discoverable devices, once a BD_ADDR has been discovered for a non-discoverable device, this can be passed off the hcitool which can now determine more information on the device, using the command **hcitool info <bdaddr>**. (Note: as already described, this is similar to the functionality the extended scan should be performing, but does not actually seem to be doing at present). "If the RSSI indication falls below a given level, the Bluetooth power level can be increased to bring the RSSI level up to an accepted level" (Poole, 2007). While this may not be accurate enough for triangulation purposes, it would certainly help to determine if a target was getting closer or further away from the tracker device.

6.7 Overall Conclusion

In combination with the right Bluetooth dongle, the Ubertooth One provides a powerful toolset to the compliance auditor's toolbox, and can offer invaluable information to any wireless vulnerability assessment.

Besides picking out the low hanging fruit of discoverable devices, its ability to identify non-discoverable devices sets it apart, which opens up a whole new category of devices that potentially need to be logged, recorded and managed.

Limited to pre-EDR versions, this apparent limitation can highlight Bluetooth traffic that runs on older (more vulnerable) versions of the Bluetooth specification, so that mitigation steps can be taken "Organisations should migrate BR legacy devices to hardware supporting EDR to mitigate Ubertooth packet capture eavesdropping threats" (Wright & Cache, 2015, pp. 253-254).

Using such guidelines as those offered by NIST (Appendix B), or attempting to meet the wireless requirements as set out by the Payment Card Industry Data Security Standards, the Ubertooth One offers security auditors a low cost tool capable of creating asset inventories, while also performing asset discovery, which could be potentially integrated into a SIEM infrastructure, to provide another layer of security to any defence in depth strategy. The number of devices containing Bluetooth chipsets will continue to rise and this area of research will become more and more relevant as security and compliance auditors attempt to stem the tidal wave of vulnerabilities brought by the *Bring Your Own Device* (BYOD) and *Internet of Things* (IoT) phenomena.

References

Cache, J., Wright, J. & Liu, V., 2010. *Hacking Exposed Wireless: Wireless security secrets & solutions.* New York: McGraw Hill.

Cisco, 2014. *Wireless Networking and PCI Compliance Networking Solutions White Paper,* San Jose: CISCO.

Coffer, J., 2013. *MAC Find.* [Online]
Available at: http://coffer.com/mac_find/
[Accessed 25 May 2015].

Davies, A., 2013. *Bluetooth Sniffing - Why bother?.* [Online]
Available at: https://penturalabs.wordpress.com/2013/09/04/bluetooth-sniffing-why-bother/
[Accessed 12 November 2014].

Davies, A., 2013. *Ubertooth - Open-Source Bluetooth Sniffing.* [Online]
Available at: http://penturalabs.wordpress.com/2013/09/01/ubertooth-open-source-bluetooth-sniffing/
[Accessed 12 November 204].

Davies, A., 2014. *Ubertooth - Bluetooth Sniffing Updated for 2014!.* [Online]
Available at: http://penturalabs.wordpress.com/2014/02/20/ubertooth-updated-for-2014/
[Accessed 12 November 2014].

Gelbstein, E., 2014. Imperfect Technologies and Digital Hygiene. *ISACA Journal,* Volume 5, pp. 17-19.

Gupta, N., 2013. *Inside Bluetooth Low Energy.* Boston: Artech House.

Harris, S., 2013. *CISSP Exam Guide.* 6th ed. s.l.:McGraw-Hill Education.

Holeman, R., 2013. *Passive Aggressive Bluetooth Scanning with Python.* [Online]
Available at: http://www.hackgnar.com/2014/03/passive-aggressive-bluetooth-scanning.html
[Accessed 22 March 2015].

Huang, A. S. & Rudolph, L., 2007. *Bluetooth Essentials for Programmers.* New York: Cambridge.

Kohlenberg, T. et al., IDS and IPS Toolkit. *Snort.* 1st ed. 2007: Syngress.

Moser, M., 2007. *Busting the Bluetooth Myth - Getting RAW Access.* [Online]
Available at: www.remote-exploit.org/content/busting_bluetooth_myth.pdf
[Accessed 23 October 2014].

Nardi, T., 2010. Bluetooth Hacking Primer. *2600: The Hackers Quarterly,* April.27(1).

Nardi, T., 2012. Bluetooth Hunter's Guide. *2600: The Hacker Quarterly,* July.29(2).

Nardi, T., 2014. *Back to the Future: Pwn Pad Review.* [Online]
Available at: http://www.thepowerbase.com/2013/04/back-to-the-future-pwn-pad-review/
[Accessed 2 April 2015].

Ossmann, M., 2014. *Project Ubertooth.* [Online]
Available at: http://ubertooth.sourceforge.net/
[Accessed 16 October 2014].

Ossmann, M., 2014. *Project Ubertooth: Discovering the Bluetooth UAP.* [Online]
Available at: http://ubertooth.blogspot.ie/2014/06/discovering-bluetooth-uap.html
[Accessed 23 October 2014].

Padgette, J., Scarfone, K. & Chen, L., 2012. *Guide to Bluetooth Security,* s.l.: U.S. Department of Commerce.

Peter, 2011. *Ubertooth is so Sweet, it Hurts!.* [Online]
Available at: https://www.xecurepla.net/ubertooth-is-so-sweet-it-hurts/
[Accessed 17 October 2014].

Poole, I., 2007. *Bluetooth Technology Tutorial.* [Online]
Available at: http://www.radio-electronics.com/info/wireless/bluetooth/bluetooth_overview.php
[Accessed 23 November 2014].

Ryan, M., 2014. *BLE Fun With Ubertooth: Sniffing Bluetooth Smart and Cracking Its Crypto.* [Online]
Available at:
http://blog.lacklustre.net/posts/BLE_Fun_With_Ubertooth:_Sniffing_Bluetooth_Smart_and_Cracking_Its_Crypto/
[Accessed 23 October 2014].

Scarfone, K. & Mell, P., 2007. *Guide to Intrusion Detection and Preventions Systems (IDPS),* s.l.: U.S. Department of Commerce.

SIG, B., 2015. *Host Controller Interface.* [Online]
Available at: https://www.bluetooth.org/en-us/specification/assigned-numbers/host-controller-interface
[Accessed 25 May 2015].

SIG, B., 2015. *Link Manager.* [Online]
Available at: https://www.bluetooth.org/en-us/specification/assigned-numbers/link-manager
[Accessed 25 May 2015].

Souppaya, M. & Scarfone, K., 2013. *Guidelines for Managing the Security of Mobile Devices in the Enterprise,* s.l.: U.S. Department of Commerce.

Spill, D., 2012. *Discovering Bluetooth Devices.* [Online]
Available at: http://ubertooth.blogspot.ie/
[Accessed 1 April 2015].

Spill, D., 2015. *Re: quick ubertooth-scan question.* s.l.:s.n.

Tipton, H. F., 2010. *Official (ISC)2 Guide to the CISSP CBK.* 2nd ed. s.l.:CRC Press.

Walker, M., 2012. *Certified Ethical Hacker.* s.l.:McGraw Hill.

Wireless Special Interest Group; PCI Security Standards Council, 2011. *Information Supplement: PCI DSS Wireless Guidelines,* s.l.: PCI Standards Council.

Wright, J., 2007. *Five Wireless Threats You May Not Know.* [Online]
Available at: http://www.sans.edu/research/security-laboratory/article/wireless-security-1
[Accessed 14 January 2015].

Wright, J., 2011. *The Bluetooth Dilemma.* [Online]
Available at: http://pen-testing.sans.org/blog/2011/10/20/the-bluetooth-dilemma
[Accessed 23 October 2014].

Wright, J., 2015. *BNAP, BNAP.* [Online]
Available at: http://bnap.opensecurityresearch.com/
[Accessed 17 May 2014].

Wright, J. & Cache, J., 2015. *Hacking Exposed Wireless.* 3rd ed. New York: McGraw Hill Education.

Appendix A – Bluetooth Profiles

1. Advanced Audio Distribution Profile (A2DP)
2. Attribute Profile (ATT)
3. Audio/Video Remote Control Profile (AVRCP)
4. Basic Imaging Profile (BIP)
5. Basic Printing Profile (BPP)
6. Common ISDN Access Profile (CIP)
7. Cordless Telephony Profile (CTP)
8. Device ID Profile (DIP)
9. Dial-up Networking Profile (DUN)
10. Fax Profile (FAX)
11. File Transfer Profile (FTP)
12. Generic Audio/Video Distribution Profile (GAVDP)
13. Generic Access Profile (GAP)
14. Generic Attribute Profile (GATT)
15. Generic Object Exchange Profile (GOEP)
16. Hard Copy Cable Replacement Profile (HCRP)
17. Health Device Profile (HDP)
18. Hands-Free Profile (HFP)
19. Human Interface Device Profile (HID)
20. Headset Profile (HSP)
21. Intercom Profile (ICP)
22. LAN Access Profile (LAP)
23. Message Access Profile (MAP)
24. OBject EXchange (OBEX)
25. Object Push Profile (OPP)
26. Personal Area Networking Profile (PAN)
27. Phone Book Access Profile (PBAP, PBA)
28. Proximity Profile (PXP)
29. Serial Port Profile (SPP)
30. Service Discovery Application Profile (SDAP)
31. SIM Access Profile (SAP, SIM, rSAP)
32. Synchronisation Profile (SYNCH)
33. Synchronisation Mark-up Language Profile (SyncML)
34. Video Distribution Profile (VDP)
35. Wireless Application Protocol Bearer (WAPB)

Appendix B – Bluetooth Security Checklist (Padgette, et al., 2012)

	Security Recommendations	Security Need, Requirement, or Justification	Checklist		
			Recom-mended Practice	Should Consider	Status
Management Recommendations					
1	Develop an organisational wireless security policy that addresses Bluetooth technology.	A security policy is the foundation for all other countermeasures.	✓		
2	Ensure that Bluetooth users on the network are made aware of their security-related responsibilities regarding Bluetooth use.	A security awareness program helps users to follow practices that help prevent security incidents.	✓		
3	Perform comprehensive security assessments at regular intervals to fully understand the organisation's Bluetooth security posture.	Assessments help identify Bluetooth devices being used within the organisation and help ensure the wireless security policy is being followed.	✓		
4	Ensure that wireless devices and networks involving Bluetooth technology are fully understood from an architecture perspective and documented accordingly.	Bluetooth-enabled devices can contain various networking technologies and interfaces, allowing connections to local and wide area networks. An organisation should understand the overall connectivity of each device to identify possible risks and vulnerabilities. These risks and vulnerabilities can then be addressed in the wireless security policy.	✓		
5	Provide users with a list of precautionary measures they should take to better protect handheld Bluetooth devices from theft.	The organisation and its employees are responsible for its wireless technology components because theft of those components could lead to malicious activities against the organisation's information system resources.	✓		

6	Maintain a complete inventory of all Bluetooth-enabled wireless devices and addresses (BD_ADDRs).	A complete inventory list of Bluetooth-enabled wireless devices can be referenced when conducting an audit that searches for unauthorised use of wireless technologies.		✓	
Technical Recommendations					
7	Change the default settings of the Bluetooth device to reflect the organisation's security policy.	Because default settings are generally not secure, a careful review of those settings should be performed to ensure that they comply with the organisational security policy. For example, the default device name should usually be changed to be non-descriptive (i.e., so that it does not reveal the platform type).	✓		
8	Set Bluetooth devices to the lowest necessary and sufficient power level so that transmissions remain within the secure perimeter of the organisation.	Setting Bluetooth devices to the lowest necessary and sufficient power level ensures a secure range of access to authorised users. The use of Class 1 devices, as well as external amplifiers or high-gain antennas, should be avoided because of their extended range.	✓		
9	Choose PIN codes that are sufficiently random, long and private. Avoid static and weak PINs, such as all zeroes.	PIN codes should be random so that malicious users cannot easily guess them. Longer PIN codes are more resistant to brute force attacks. For Bluetooth v2.0 (or earlier) devices, an eight-character alphanumeric PIN should be used, if possible. The use of a fixed PIN is not acceptable.	✓		
10	Ensure that link keys are not based on unit keys.	The use of shared unit keys can lead to successful spoofing, MITM, and eavesdropping attacks. The use of unit keys for security was deprecated in Bluetooth v1.2.	✓		
11	For v2.1 and later devices using SSP, avoid using the "Just Works" association model.	The "Just Works" association model does not provide MITM protection. Devices that only support Just Works (e.g., devices	✓		

	The device must verify that an authenticated link key was generated during pairing.	that have no input/output capability) should not be procured if similarly qualified devices that support one of the other association models (i.e., Numeric Comparison, OOB, or Passkey Entry) are available.			
12	For v2.1 and later devices using SSP, random and unique passkeys must be used for each pairing based on the Passkey Entry association model.	If a static passkey is used for multiple pairings, the MITM protection provided by the Passkey Entry association model is reduced.	✓		
13	A Bluetooth v2.1 or later device using Security Mode 4 must fall back to Security Mode 3 for backward compatibility with v2.0 and earlier devices (i.e., for devices that do not support Security Mode 4).	The Bluetooth specifications allow a v2.1 device to fall back to any Security Mode for backward compatibility. This allows the option of falling back to Security Modes 1-3. As discussed earlier, Security Mode 3 provides the best security.	✓		
14	LE devices and services should use Security Mode 1 Level 3 whenever possible. LE Security Mode 1 Level 3 provides the highest security available for LE devices	Other LE security modes allow unauthenticated pairing and/or no encryption.	✓		
15	Unneeded and unapproved service and profiles should be disabled.[1]	Many Bluetooth stacks are designed to support multiple profiles and associated services. The Bluetooth stack on a device should be locked down to ensure only required and approved profiles and services are available for use.	✓		
16	Bluetooth devices should be configured by default as undiscoverable and remain undiscoverable except as needed for pairing.	This prevents visibility to other Bluetooth devices except when discovery is absolutely required. In addition, the default Bluetooth device names sent during discovery should be changed to non-	✓		

[1] Derived from requirement 1.4 in the DoD Bluetooth Peripheral Security Requirements (16 July 2010), available at http://iase.disa.mil/stigs/downloads/pdf/dod_bluetooth_requirements_spec_20100716.pdf

		identifying values.			
17	Invoke link encryption for all Bluetooth connections.	Link encryption should be used to secure all data transmissions during a Bluetooth connection; otherwise, transmitted data are vulnerable to eavesdropping.	✓		
18	If multi-hop wireless communication is being used, ensure that encryption is enabled on every link in the communication chain.	One unsecured link results in compromising the entire communication chain.	✓		
19	Ensure device mutual authentication is performed for all connections.	Mutual authentication is required to provide verification that all devices on the network are legitimate.	✓		
20	Enable encryption for all broadcast transmissions (Encryption Mode 3).	Broadcast transmissions secured by link encryption provide a layer of security that protects these transmissions from user interception for malicious purposes.	✓		
21	Configure encryption key sizes to the maximum allowable (128-bit).	Using maximum allowable key sizes provides protection from brute force attacks.	✓		
22	Use application-level authentication and encryption atop the Bluetooth stack for sensitive data communication.	Bluetooth devices can access link keys from memory and automatically connect with previously paired devices. Incorporating application-level software that implements authentication and encryption will add an extra layer of security. Passwords and other authentication mechanisms, such as biometrics and smart cards, can be used to provide user authentication for Bluetooth devices. Employing higher layer encryption (particularly FIPS 140 validated) over the native encryption will further protect the		✓	

		data in transit.			
23	Deploy user authentication overlays such as biometrics, smart cards, two-factor authentication, or public key infrastructure (PKI).	Implementing strong authentication mechanisms can minimise the vulnerabilities associated with passwords and PINs.		✓	
Operational Recommendations					
24	Ensure that Bluetooth capabilities are disabled when they are not in use.	Bluetooth capabilities should be disabled on all Bluetooth devices, except when the user explicitly enables Bluetooth to establish a connection. This minimises exposure to potential malicious activities. For devices that do not support disabling Bluetooth (e.g., headsets), the entire device should be shut off when not in use.	✓		
25	Perform pairing as infrequently as possible, ideally in a secure area where attackers cannot realistically observe the passkey entry and intercept Bluetooth pairing messages. (Note: A "secure area" is defined as a non-public area that is indoors away from windows in locations with physical access controls.) Users should not respond to any messages requesting a PIN, unless the user has initiated a pairing and is certain the PIN request is being sent by one of the user's devices.[2]	Pairing is a vital security function and requires that users maintain a security awareness of possible eavesdroppers. If an attacker can capture the transmitted frames associated with pairing, determining the link key is straightforward for pre-v2.1 and v4.0 devices since security is solely dependent on PIN entropy and length. This recommendation also applies to v2.1/3.0 devices, although similar eavesdropping attacks against SSP have not yet been documented.	✓		
26	A BR/EDR service-level security mode (i.e., Security Mode 2 or 4) should only be used in a controlled and well-understood environment.	Security Mode 3 provides link-level security prior to link establishment, while Security Modes 2 and 4 allow link-level connections before any authentication or encryption is established. NIST highly recommends that devices use	✓		

[2] Derived from requirement 4.1.5 in the DoD Bluetooth Peripheral Security Requirements (16 July 2010), available at http://iase.disa.mil/stigs/downloads/pdf/dod_bluetooth_requirements_spec_20100716.pdf

		Security Mode 3.			
27	Ensure that portable devices with Bluetooth interfaces are configured with a password.	This helps prevent unauthorised access if the device is lost or stolen.	✓		
28	In the event a Bluetooth device is lost or stolen, users should immediately delete the missing device from the paired device lists of all other Bluetooth devices.	This policy will prevent an attacker from using the lost or stolen device to access another Bluetooth device owned by the user(s).	✓		
29	Install antivirus software on Bluetooth-enabled hosts that support such host-based security software.	Antivirus software should be installed to ensure that known malware is not introduced to the Bluetooth network.	✓		
30	Fully test and regularly deploy Bluetooth software and firmware patches and upgrades.	Newly discovered security vulnerabilities of vendor products should be patched to prevent malicious and inadvertent exploits. Patches should be fully tested before implementation to confirm that they are effective.	✓		
31	Users should not accept transmissions of any kind from unknown or suspicious devices. These types of transmissions include messages, files, and images.	With the increase in the number of Bluetooth-enabled devices, it is important that users only establish connections with other trusted devices and only accept content from these trusted devices	✓		
32	Fully understand the impacts of deploying any security feature or product prior to deployment.	To ensure a successful deployment, an organisation should fully understand the technical, security, operational, and personnel requirements prior to implementation.	✓		
33	Designate an individual to track the progress of Bluetooth security products and standards (perhaps via the Bluetooth SIG) and the threats and vulnerabilities with the technology.	An individual designated to track the latest technology enhancements, standards (perhaps via Bluetooth SIG), and risks will help to ensure the continued secure use of Bluetooth.		✓	

Appendix C – 5 devices attached hciconfig output

```
root@kali:~# hciconfig
hci2:	Type: BR/EDR  Bus: USB
	BD Address: 00:50:C2:7F:47:80  ACL MTU: 310:10  SCO MTU: 64:8
	DOWN 
	RX bytes:859 acl:0 sco:0 events:34 errors:0
	TX bytes:388 acl:0 sco:0 commands:31 errors:0

hci4:	Type: BR/EDR  Bus: USB
	BD Address: 00:13:10:5D:3F:55  ACL MTU: 377:10  SCO MTU: 64:8
	DOWN 
	RX bytes:1227 acl:0 sco:0 events:43 errors:0
	TX bytes:436 acl:0 sco:0 commands:43 errors:0

hci3:	Type: BR/EDR  Bus: USB
	BD Address: 00:01:95:21:C4:95  ACL MTU: 310:10  SCO MTU: 64:8
	DOWN 
	RX bytes:1471 acl:0 sco:0 events:71 errors:0
	TX bytes:1270 acl:0 sco:0 commands:71 errors:0

hci1:	Type: BR/EDR  Bus: USB
	BD Address: 00:0C:41:E2:77:7B  ACL MTU: 192:8  SCO MTU: 64:8
	DOWN 
	RX bytes:1025 acl:0 sco:0 events:33 errors:0
	TX bytes:376 acl:0 sco:0 commands:33 errors:0

hci0:	Type: BR/EDR  Bus: USB
	BD Address: 78:DD:08:B2:DE:4C  ACL MTU: 1021:8  SCO MTU: 64:1
	DOWN 
	RX bytes:10117 acl:0 sco:0 events:471 errors:0
      TX bytes:5407 acl:0 sco:0 commands:474 errors:1
```

Appendix D – ubertooth-scan help output

```
root@kali:~# Ubertooth-scan -h
ubertooth-scan - active(bluez) device scan and inquiry supported by Ubertooth
Usage:
        -h this Help
        -s hci Scan - perform HCI scan
        -t scan Time (seconds) - length of time to sniff packets. [Default: 20s]
        -x eXtended scan - retrieve additional information about target devices
        -b Bluetooth device (hci0)
root@kali:~#
```

Appendix E – Internal laptop device hciconfig output

```
root@kali:~#
root@kali:~# hciconfig hci1 -a
hci1:    Type: BR/EDR  Bus: USB
         BD Address: 78:DD:08:B2:DE:4C  ACL MTU: 1021:8  SCO MTU: 64:1
         UP RUNNING PSCAN
         RX bytes:2192 acl:0 sco:0 events:66 errors:0
         TX bytes:976 acl:0 sco:0 commands:66 errors:0
         Features: 0xff 0xff 0x8f 0xfe 0x9b 0xff 0x79 0x83
         Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
         Link policy: RSWITCH HOLD SNIFF PARK
         Link mode: SLAVE ACCEPT
         Name: 'kali-0'
         Class: 0x420100
         Service Classes: Networking, Telephony
         Device Class: Computer, Uncategorized
         HCI Version: 2.1 (0x4)  Revision: 0x168
         LMP Version: 2.1 (0x4)  Subversion: 0x4203
         Manufacturer: Broadcom Corporation (15)

root@kali:~# hciconfig hci1 revision
hci1:    Type: BR/EDR  Bus: USB
         BD Address: 78:DD:08:B2:DE:4C  ACL MTU: 1021:8  SCO MTU: 64:1
         Firmware 104.66 / 3
root@kali:~#
```

Appendix F – Marked Nano device hciconfig output

```
root@kali:~#
root@kali:~# hciconfig hci0 -a
hci0:	Type: BR/EDR  Bus: USB
	BD Address: 00:19:86:00:3C:65  ACL MTU: 1017:8  SCO MTU: 64:0
	UP RUNNING PSCAN
	RX bytes:819 acl:0 sco:0 events:31 errors:0
	TX bytes:381 acl:0 sco:0 commands:31 errors:0
	Features: 0xff 0xff 0x8d 0xfe 0x9b 0xf9 0x00 0x80
	Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
	Link policy: RSWITCH HOLD SNIFF PARK
	Link mode: SLAVE ACCEPT
	Name: 'kali-1'
	Class: 0x420100
	Service Classes: Networking, Telephony
	Device Class: Computer, Uncategorized
	HCI Version: 2.0 (0x3)  Revision: 0x3000
	LMP Version: 2.0 (0x3)  Subversion: 0x420b
	Manufacturer: Broadcom Corporation (15)

root@kali:~# hciconfig hci0 revision
hci0:	Type: BR/EDR  Bus: USB
	BD Address: 00:19:86:00:3C:65  ACL MTU: 1017:8  SCO MTU: 64:0
	Firmware 0.66 / 11
root@kali:~#
```

Appendix G – Unmarked Nano device hciconfig output

```
root@kali:~#
root@kali:~# hciconfig hci0 -a
hci0:    Type: BR/EDR  Bus: USB
         BD Address: 00:15:83:0C:BF:EB  ACL MTU: 339:8  SCO MTU: 128:2
         UP RUNNING PSCAN
         RX bytes:948 acl:0 sco:0 events:34 errors:0
         TX bytes:372 acl:0 sco:0 commands:27 errors:0
         Features: 0xff 0x3e 0x85 0x30 0x18 0x18 0x00 0x00
         Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
         Link policy: RSWITCH HOLD SNIFF
         Link mode: SLAVE ACCEPT
         Name: 'kali-1'
         Class: 0x000104
         Service Classes: Unspecified
         Device Class: Computer, Desktop workstation
         HCI Version: 2.0 (0x3)  Revision: 0xc5c
         LMP Version: 2.0 (0x3)  Subversion: 0xc5c
         Manufacturer: Cambridge Silicon Radio (10)

root@kali:~# hciconfig hci0 revision
hci0:    Type: BR/EDR  Bus: USB
         BD Address: 00:15:83:0C:BF:EB  ACL MTU: 339:8  SCO MTU: 128:2
         Build 0
         Chip version: BlueCore01a
         Max key size: 0 bit
         SCO mapping:  PCM
root@kali:~#
```

Appendix H – Linksys unmodified hciconfig output

```
root@kali:~#
root@kali:~# hciconfig hci0 -a
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:0C:41:E2:77:7B  ACL MTU: 192:8  SCO MTU: 64:8
        UP RUNNING PSCAN
        RX bytes:672 acl:0 sco:0 events:22 errors:0
        TX bytes:337 acl:0 sco:0 commands:21 errors:0
        Features: 0xff 0xff 0x0f 0x00 0x00 0x00 0x00 0x00
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy:
        Link mode: SLAVE ACCEPT
        Name: 'kali-1'
        Class: 0x420100
        Service Classes: Networking, Telephony
        Device Class: Computer, Uncategorized
        HCI Version: 1.1 (0x1)  Revision: 0x20d
        LMP Version: 1.1 (0x1)  Subversion: 0x20d
        Manufacturer: Cambridge Silicon Radio (10)

root@kali:~# hciconfig hci0 revision
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:0C:41:E2:77:7B  ACL MTU: 192:8  SCO MTU: 64:8
        HCI 16.4
        Chip version: BlueCore02-External
        Max key size: 56 bit
        SCO mapping:  HCI
root@kali:~#
```

Appendix I – Linksys modified hciconfig output

```
root@kali:~#
root@kali:~# hciconfig hci0 -a
hci0:	Type: BR/EDR  Bus: USB
	BD Address: 00:13:10:5D:3F:55  ACL MTU: 377:10  SCO MTU: 64:8
	UP RUNNING PSCAN
	RX bytes:767 acl:0 sco:0 events:26 errors:0
	TX bytes:367 acl:0 sco:0 commands:26 errors:0
	Features: 0xff 0xfe 0x0d 0x38 0x08 0x08 0x00 0x00
	Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
	Link policy:
	Link mode: SLAVE ACCEPT
	Name: 'kali-1'
	Class: 0x420100
	Service Classes: Networking, Telephony
	Device Class: Computer, Uncategorized
	HCI Version: 1.2 (0x2)  Revision: 0x0
	LMP Version: 1.2 (0x2)  Subversion: 0x309
	Manufacturer: Broadcom Corporation (15)

root@kali:~# hciconfig hci0 revision
hci0:	Type: BR/EDR  Bus: USB
	BD Address: 00:13:10:5D:3F:55  ACL MTU: 377:10  SCO MTU: 64:8
	Firmware 0.3 / 9
root@kali:~#
```

Appendix J – Aircable device hciconfig output

```
root@kali:~#
root@kali:~# hciconfig hci0 -a
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:50:C2:7F:47:80  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING PSCAN
        RX bytes:859 acl:0 sco:0 events:34 errors:0
        TX bytes:388 acl:0 sco:0 commands:31 errors:0
        Features: 0xff 0xff 0x8f 0xf8 0x1b 0xf8 0x00 0x80
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH HOLD SNIFF PARK
        Link mode: SLAVE ACCEPT
        Name: 'kali-1'
        Class: 0x420100
        Service Classes: Networking, Telephony
        Device Class: Computer, Uncategorized
        HCI Version: 2.0 (0x3)  Revision: 0x10b7
        LMP Version: 2.0 (0x3)  Subversion: 0x10b7
        Manufacturer: Cambridge Silicon Radio (10)

root@kali:~# hciconfig hci0 revision
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:50:C2:7F:47:80  ACL MTU: 310:10  SCO MTU: 64:8
        Unified 22b
        Chip version: BlueCore4-External
        Max key size: 56 bit
        SCO mapping:  HCI
root@kali:~#
```

Appendix K – SENA device hciconfig output

```
root@kali:~#
root@kali:~# hciconfig hci0 -a
hci0:	Type: BR/EDR  Bus: USB
	BD Address: 00:01:95:21:C4:95  ACL MTU: 310:10  SCO MTU: 64:8
	UP RUNNING PSCAN
	RX bytes:4305 acl:0 sco:0 events:149 errors:0
	TX bytes:2263 acl:0 sco:0 commands:148 errors:0
	Features: 0xff 0xff 0x8f 0xfe 0xdb 0xff 0x5b 0x87
	Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
	Link policy: RSWITCH HOLD SNIFF PARK
	Link mode: SLAVE ACCEPT
	Name: 'kali-1'
	Class: 0x420100
	Service Classes: Networking, Telephony
	Device Class: Computer, Uncategorized
	HCI Version: 4.0 (0x6)  Revision: 0x2031
	LMP Version: 4.0 (0x6)  Subversion: 0x2031
	Manufacturer: Cambridge Silicon Radio (10)

root@kali:~# hciconfig hci0 revision
hci0:	Type: BR/EDR  Bus: USB
	BD Address: 00:01:95:21:C4:95  ACL MTU: 310:10  SCO MTU: 64:8
	Build 8241
	Chip version: Unknown
	Max key size: 128 bit
	SCO mapping:  HCI
root@kali:~#
```

Appendix L – Shell script (version 1)

scan1.bash

```
#!/bin/bash

echo "run hciconfig command - for reference purposes"
hciconfig

echo "Let's bring down all the devices, and bring them up one at a time"

for i in hci0 hci1 hci2 hci3 hci4; do
        echo hciconfig $i down
             hciconfig $i down
        echo
done

echo "run hciconfig command - for reference purposes"
hciconfig
echo

for j in hci0 hci1 hci2 hci3 hci4; do

        echo "Let's bring $j up"
        echo hciconfig $j up
             hciconfig $j up
        echo

        echo "run hciconfig command - for reference purposes"
        hciconfig
        echo

        echo "$j - ubertooth-scan"
        echo ubertooth-scan -b $j -t 1800
             ubertooth-scan -b $j -t 1800
        echo

        echo "$j - hci scan"
        echo ubertooth-scan -b $j -t 1800 -s
             ubertooth-scan -b $j -t 1800 -s
        echo


        echo "$j - extended scan"
        echo ubertooth-scan -b $j -t 1800 -x
             ubertooth-scan -b $j -t 1800 -x
        echo

        echo "Let's bring $j down"
        echo hciconfig $j down
             hciconfig $j down
        echo

done
```

Appendix M – Shell script (version 2)

Scan2.bash

```
#!/bin/bash

echo "run hciconfig command - for reference purposes"
hciconfig

echo "Let's bring down all the devices, and bring them up one at a time"

for i in hci0 hci1 hci2 hci3 hci4; do
        echo hciconfig $i down
             hciconfig $i down
        echo
done

echo "run hciconfig command - for reference purposes"
hciconfig
echo

for j in hci0 hci1 hci2 hci3 hci4; do

        echo "Let's bring $j up"
        echo hciconfig $j up
             hciconfig $j up
        echo

        echo "run hciconfig command - for reference purposes"
        hciconfig
        echo

        echo "$j - ubertooth-scan"
        echo ubertooth-scan -b $j -t 900
             ubertooth-scan -b $j -t 900
        echo

        echo "$j - hci scan"
        echo ubertooth-scan -b $j -t 900 -s
             ubertooth-scan -b $j -t 900 -s
        echo


        echo "$j - extended scan"
        echo ubertooth-scan -b $j -t 900 -x
             ubertooth-scan -b $j -t 900 -x
        echo

        echo "Let's bring $j down"
        echo hciconfig $j down
             hciconfig $j down
        echo

done
```

Appendix N – Shell scripts (version 3)

scan_master.bash

```
#!/bin/bash

echo hciconfig
hciconfig

echo bring devices down
for i in hci0 hci1 hci2 hci3 hci4; do
        echo hciconfig $i down
        hciconfig i$i down
done
echo hciconfig
hciconfig

echo bring devices up
for j in hci0 hci1 hci2 hci3 hci4; do
        echo hciconfig $j up
        hciconfig $j up

        echo hciconfig
        hciconfig

        echo run scripts here
        ./scan_slave1.bash $j
        ./scan_slave2.bash $j
        ./scan_slave3.bash $j

        echo hciconfig $j down
        hciconfig $j down

        echo hciconfig
        hciconfig
done
```

scan_slave1.bash

```
#!/bin/bash

echi scan_slave1.bash starting
echo device - $1

echo ubertooth-scan -b $1 -t 900
     ubertooth-scan -b $1 -t 900

echo scan_slave1.bash completed
```

scan_slave2.bash

```
#!/bin/bash

echi scan_slave2.bash starting
echo device - $1

echo ubertooth-scan -b $1 -t 900 -s
     ubertooth-scan -b $1 -t 900 -s

echo scan_slave2.bash completed
```

scan_slave3.bash

```
#!/bin/bash

echi scan_slave3.bash starting
echo device - $1

echo ubertooth-scan -b $1 -t 900 -x
     ubertooth-scan -b $1 -t 900 -x

echo scan_slave3.bash completed
```

Appendix O – Results processing – perl script

bluetooth.pl

```
#!/usr/bin/perl
use warnings; use strict;
use constant DEBUGGING => 0;
print scalar(localtime(time())) . "\tStarting $0\n";
my $output   = $ARGV[0] || die "instance missing output type\n";
my $filename = $ARGV[1] || die "instance missing filename\n";
print "FILENAME: $filename\n";

open (INPUT_FILE, "$filename") or die $!;
foreach my $line (<INPUT_FILE>){
    chomp $line;
    next unless ($line=~m/^systime/);
    print $line . "\n" if (DEBUGGING > 0);

   if
($line=~m/^systime=(\d+)\sch=\s?(\d+)\sLAP=(\w+)\serr=(\d+)\sclk100ns=(\d+)\sclk1=(\d
+)\ss=(-\d+)\sn=(-\d+)\ssnr=(\d+)/){
        my $systime = $1;
        my $ch = $2;
        my $LAP = $3;
        my $err = $4;
        my $clk100ns = $5;
        my $clk1 = $6;
        my $s = $7;
        my $n = $8;
        my $snr = $9;
        if (DEBUGGING > 0){
            print "\$systime >>>$systime<<<\n";
            print "\$ch >>>$ch<<<\n";
            print "\$LAP >>>$LAP<<<\n"; print "\$err >>>$err<<<\n";
            print "\$clk100ns >>>$clk100ns<<<\n";
            print "\$clk1 >>>$clk1<<<\n";
            print "\$s >>>$s<<<\n"; print "\$n >>>$n<<<\n";
            print "\$snr >>>$snr<<<\n";
        }
        if ($output eq "SYSLOG"){
            my $new_systime = localtime($systime);
            print "\$new_systime - New Bluetooth Device Found - $LAP\n";
        }
        elsif  ($output eq "EXCEL"){
            print "$systime,'$LAP\n";
        }
    }
}
close (INPUT_FILE);
print scalar(localtime(time())) . "\tFinished $0\n";
exit (0);
__END__
```

Appendix P – Results processing – batch file

run.bat

```
rem hci0
perl bluetooth.pl EXCEL "laptop1.txt" >laptop1.csv
perl bluetooth.pl EXCEL "laptop2.txt" >laptop2.csv
perl bluetooth.pl EXCEL "laptop3.txt" >laptop3.csv
rem hci1
perl bluetooth.pl EXCEL "linksys1.txt" >linksys1.csv
perl bluetooth.pl EXCEL "linksys2.txt" >linksys2.csv
perl bluetooth.pl EXCEL "linksys3.txt" >linksys3.csv
rem hci2
perl bluetooth.pl EXCEL "aircable1.txt" >aircable1.csv
perl bluetooth.pl EXCEL "aircable2.txt" >aircable2.csv
perl bluetooth.pl EXCEL "aircable3.txt" >aircable3.csv
rem hci3
perl bluetooth.pl EXCEL "sena1.txt" >sena1.csv
perl bluetooth.pl EXCEL "sena2.txt" >sena2.csv
perl bluetooth.pl EXCEL "sena3.txt" >sena3.csv
rem hci4
perl bluetooth.pl EXCEL "linksys_m1.txt" >linksys_m1.csv
perl bluetooth.pl EXCEL "linksys_m2.txt" >linksys_m2.csv
perl bluetooth.pl EXCEL "linksys_m3.txt" >linksys_m3.csv
```