6.858: Hacking Bluetooth

Elaina Chai
echai@mit.edu

Ben Deardorff
bendorff@mit.edu

Cathy Wu
cathywu@mit.edu

09 December 2012

Abstract

After learning about and analyzing the security of Bluetooth, it was clear to us that Bluetooth sniffing tools are still substandard compared to those available for sniffing other types of wireless traffic like WiFi. This makes it harder for hackers to develop exploits for Bluetooth devices but also makes it more difficult for security researchers to realistically evaluate Bluetooth security. We decided that the best way to address this problem is to continue development of the software for the Ubertooth module, currently the most cost effective hardware device for sniffing Bluetooth packets. In this paper, we highlight the fact that Bluetooth is a widespread technology with real privacy and security implications. Furthermore, we explore the current capabilities of using inexpensive open source software and hardware to examine data from arbitrary Bluetooth devices. We have also implemented piconet following in the Kismet-Ubertooth plugin, making it an even more effective tool for future researchers in this area. Our implementation can be found at https://github.com/cathywu/6858-kismet-ubertooth.

1 Introduction

Bluetooth, since its inception in 1998, has become one of the most widely used short-range wireless protocols and has quietly become a part of our everyday lives. It is heralded for its convenience in connecting and exchanging information between devices such as cell phone headsets, mobile phones, telephones, laptops, personal computers, printers, Global Positioning System (GPS) receivers, digital cameras, video game consoles, and even faxes. Unfortunately, Bluetooth still contains a large number of security vulnerabilities despite the claims made by the Bluetooth Special Interest Group.

Despite known vulnerabilities, demonstrated hacks, and the decreasing cost of hacking Bluetooth, the number of Bluetooth devices manufactured today continues to grow. Currently, about 2 billion Bluetooth enabled devices are shipped each year, with mobile phones make up the bulk of Bluetooth shipment numbers.

In this paper we highlight the fact that Bluetooth is a widespread technology with real privacy and security implications. Furthermore, we explore the current capabilities of using inexpensive open source software and hardware to examine data from arbitrary Bluetooth devices. We have also implemented piconet following in the Kismet-Ubertooth plugin, making it an even more effective tool for future researchers in this area. Our implementation can be found at https://github.com/cathywu/6858-kismet-ubertooth.

In Section 2, we introduce the Bluetooth protocol. In Section 3, we discuss Bluetooth security, including the security model, weaknesses, barriers to hacking, and existing hardware sniffing tools. In Section 4 and 5, we discuss the tools we used and our implementation. Finally, we discuss future work in Section 6, and we conclude in Section 7.

2 Bluetooth protocol

2.1 Pairing

Originally designed to be a humble cable replacement technology, Bluetooth is now a wireless technology standard for exchanging data over short distances. Slave devices (e.g.

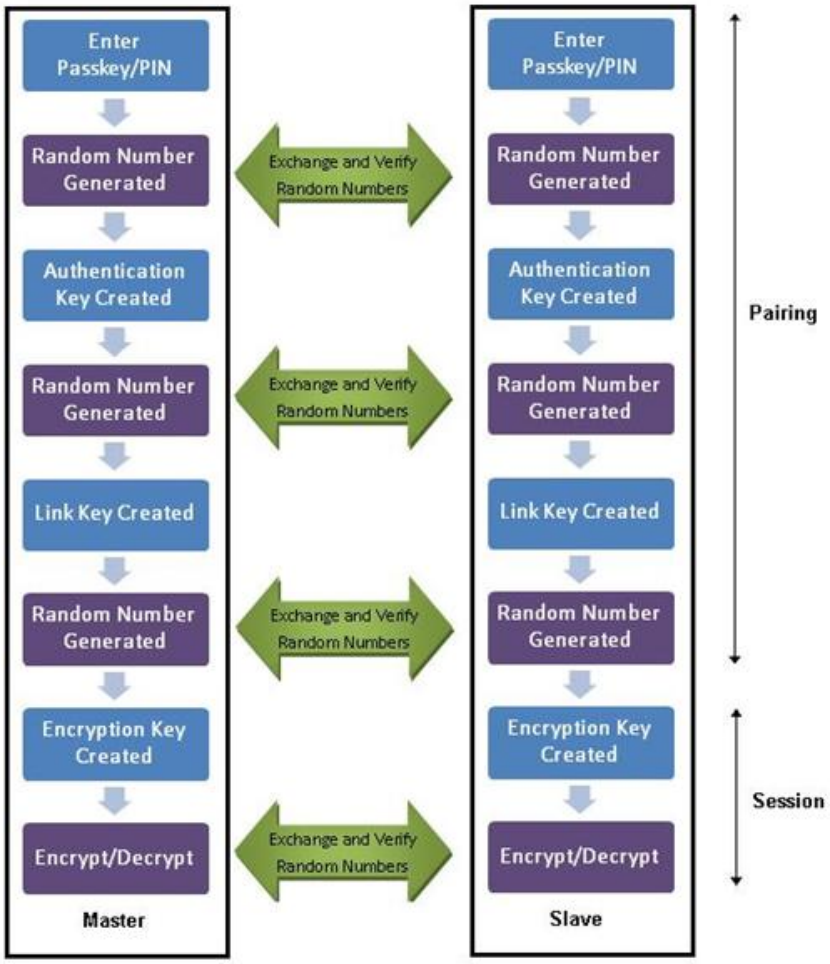

Figure 1: Pairing process.

headset, keyboard) connect with master devices (e.g. phone, laptop) by a pairing process as shown in Figure 1, in which device IDs are passed and a number of keys are generated. The pairing process usually involves some level of user interaction (e.g. entering a PIN), which is the basis for confirming the identity of the devices. Because of the user interaction, the PIN is not transmitted over the wireless channel. An initialization key is then generated from the PIN. The initialization key is used to agree upon a link key, which depends on the type of communication desired. The link key is then used to generate the encryption key.

2.2 Frequency hopping

Bluetooth is more complicated than most wireless standards in that it does not stay on any particular channel for long. In fact, 1600 times per second, Bluetooth hops between 79 1-MHz channels using a spread spectrum frequency hopping radio. A master can connect up to 7 slave devices simultaneously; paired slave devices and the master device form a piconet, wherein all the devices share a clock for synchronized communication. The devices in the piconet all use the master's information to stay in sync; the master's device ID determines the hopping pattern, and the master's clock determines the phase in hopping pattern [9]. Each device ID maps to a unique hopping pattern.

Although spread-spectrum signals are more difficult to intercept, frequency hopping is not designed to be a security mechanism. Rather, frequency-hopping spread spectrum (FHSS) is highly resistant to narrowband interference, which is where Bluetooth operates [23]. To further reduce interference from wireless LAN (Wi-Fi), which operates in the same radio band, Bluetooth also implements Adaptive Frequency Hopping (AFH). Devices operating with AFH identify "bad" channels that may cause interference and avoid them. These "bad" channels are those that are presently occupied by a WLAN transmission. WLAN devices typically do not change channels, but WLAN channels span multiple Bluetooth channels [14]. These frequency hopping schemes are an inconvenience rather than a barrier for Bluetooth hackers.

2.3 Device ID

The Bluetooth device ID (or Bluetooth Address BD_ADDR) is a unique 48-bit number used to identify each Bluetooth device, similar to Ethernet's MAC address for a computer. Unlike the MAC address, however, which is foregone for IP addresses at the higher layers of communication, the BD_ADDR is used throughout the Bluetooth protocol – for low-level radio protocols, identity, authenication, and synchronization. Within a piconet, all devices including slaves transmit using the master's device ID.

The BD_ADDR (Figure 2) consists of a 2-byte Non-significant Address Portion (NAP), a 1-byte Upper Address Portion (UAP), and a 3-byte Lower Address Portion (LAP). As the name suggests, the NAP is not used in any critical aspects of the Bluetooth protocol,

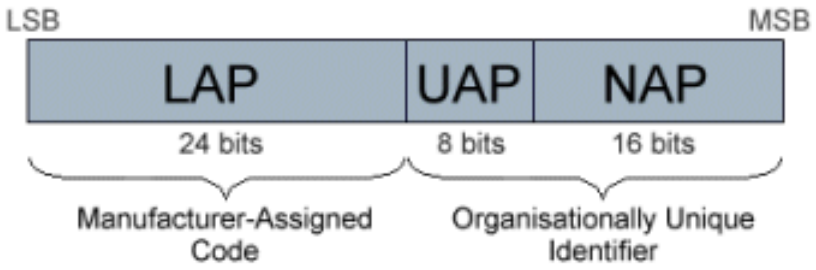

Figure 2: Bluetooth address.

except in encrypting packets. Since the BD_ADDR is so widely used, it is essential that devices maintain its privacy by keeping its BD_ADDR secret. Unfortunately, device IDs are publically available via Bluetooth inquiries for devices in Discoverable Mode. Even when not discoverable, all Bluetooth packets are sent with the LAP of the device (or of the master if part of a piconet) in cleartext. There are also straightforward methods for determining a device's UAP, by either timing the gaps between packets or extracting a cyclic redundancy check (CRC) code from a packet payload). Knowing the LAP and UAP of a device allows for passive monitoring and attacks.

2.4 Forward error correction (FEC)

Forward error correction (FEC) is a technique used for controlling errors in data transmission over unreliable or noisy communication channels. FEC uses an error-correcting code (ECC) to encode messages in a redundant way. The purpose of FEC is to reduce the number of retransmissions, not to provide security. A CRC (cyclic redundancy check) code is added to each packet and used by the receiver to decide whether or not the packet has arrived error free [1].

2.5 Data whitening

Before transmission, both the header and the payload of each packet are scrambled with a data whitening word in order to randomise the data from highly redundant patterns and to minimize DC bias in the packet. Whitening is a feature for signal transmission, not

security. The scrambling is performed prior to the FEC encoding [2].

3 Bluetooth security

3.1 Bluetooth security model

3.1.1 Security goals

Three basic security services defined by the Bluetooth specification [20]:

- **Authentication** A goal of Bluetooth is the identity verification of communicating devices based on their Bluetooth device IDs. This service provides an abort mechanism if a device cannot authenticate properly. Bluetooth does not provide native user authentication.
- **Confidentiality** Another goal of Blueooth is to maintain the privacy of users and devices. The intent is to prevent information compromise caused by eavesdropping by ensuring that only authorized devices can access and view transmitted data.
- **Authorization** A third goal of Bluetooth is a security service developed to allow the control of resources, which ensures that a device is authorized to use a service before permitting it to do so.

3.1.2 Security mechanisms

Bluetooth implements its security goals by:

- **Authorization (user inputs PIN)** In a short range wireless network, there can be no centralized, trusted party. Bluetooth offloads the question of authorizing communication between devices to the user. The user initializes the access between two devices by identifying and selecting the appropriate device by device name (e.g. ”Cathy’s Macbook Pro”, ”Keyboard01”), which maps directly to a device ID. The user is next prompted to enter a PIN that displays on one device into the other device or is instructed to follow special button presses.

- **Authentication (verify link key)** During the initalization stage, the two devices also agree upon a secret link key, derived from the PIN, and it is stored between sessions. The authentication procedure (Figure 3), then, gives one device an opportunity to verify its knowledge of the secret key, and thus its identity [3].

  The Bluetooth authentication procedure is in the form of a "challenge-response" scheme. The two devices interacting in an authentication procedure are referred to as the claimant and the verifier. The verifier is the Bluetooth device validating the identity of another device. The claimant is the device attempting to prove its identity. One of the Bluetooth devices (the claimant) attempts to reach and connect to the other (the verifier). The steps in the authentication process are the following:

  1. The claimant transmits its 48-bit address (BD ADDR) to the verifier.
  2. The verifier transmits a 128-bit random challenge (RAND) to the claimant.
  3. The verifier uses the cryptographic E1 algorithm (Bluetooth standard) to compute an authentication response using the address, link key, and random challenge as inputs. The claimant performs the same computation.
  4. The claimant returns the computed 32-bit signed response (SRES), to the verifier.
  5. The verifier compares the SRES from the claimant with the SRES that it computes.
  6. If the two SRES values are equal, the verifier will continue connection establishment.

- **Confidentiality (stream cipher encryption)** Bluetooth can encrypt the packet payload (Figure 4), using a stream cipher called E0. The cipher is stored temporarily and re-synchronizes for every payload, whereby minimizing the effectiveness of correlation attacks. As input, the E0 algorithm uses the master Bluetooth address, the master real-time clock and the encryption key. The encryption key is derived from the current link key, ciphering offset and a random number. Most Bluetooth encryption schemes use a encryption key between and 1 and 16 bytes long. The master sends the random number in plain text to the other devices before encryption is started.

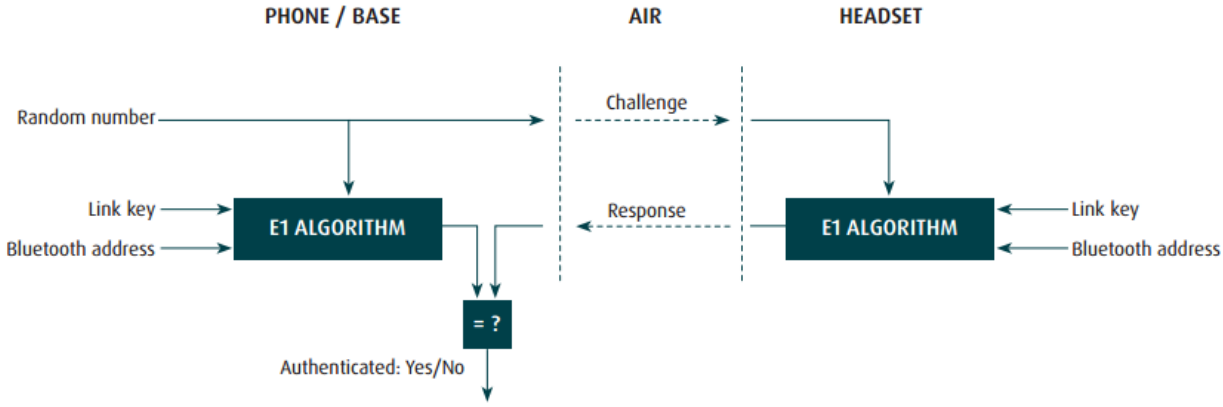

Figure 3: Authentication via verifying the link key [3].

The E0 algorithm delivers a key stream which is XOR-ed to the data that shall be encrypted. Since the cipher is symmetric, decryption is handled in the same way.

Modern Bluetooth devices implement all three security services, but in order to provide backwards compatibility and for performance considerations, each of these services are not enforced. Different levels of security are more appropriate for different applications. For example, pacemakers operating via Bluetooth should enforce more security features than a stereo that is streaming music via Bluetooth. However, these security controls are not always used appropriately.

3.2 Security assumptions

Bluetooth makes several assumptions about its security. It assumes that once a connection established, the connection between devices will be permanently secure. It also assumes that short range provides high security (an attacker is not nearby); the Bluetooth range is typically 5-30m from the device. Additionally, Bluetooth authenticates per device, assuming that all services and users on a particular device should follow the same security policy.

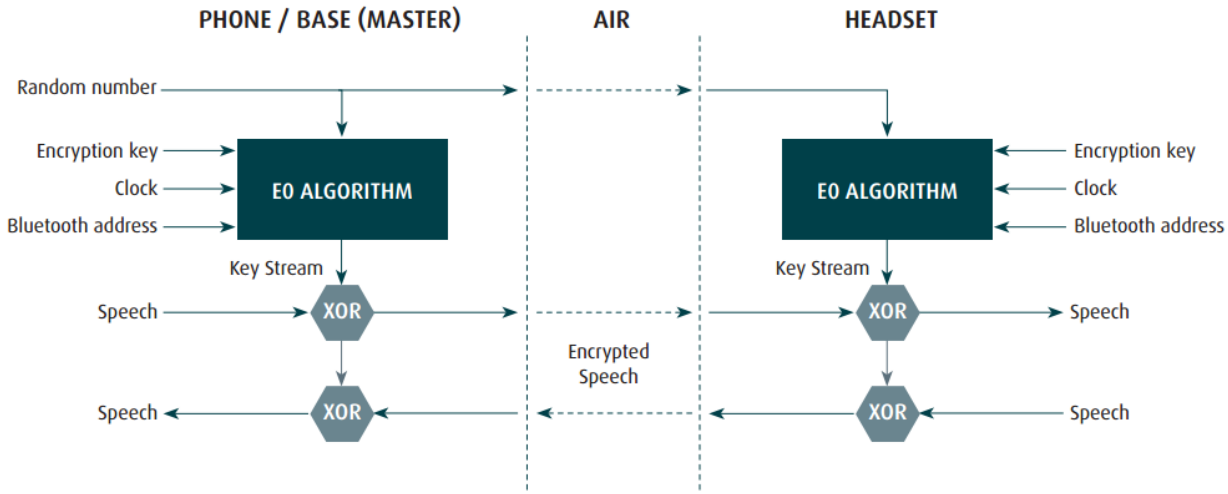

Figure 4: Confidentiality via stream cipher encryption [3].

3.3 Known Vulnerabilities and Attacks

The weakest part of the Bluetooth protocol is during the initial stages to set up the connection, before encryption is fully utilized. Because of this, a great deal of the focus of past Bluetooth attacks have been on taking advantage of the vulnerabilities in these initial stages. This is in no small part due to the weak/lack of encryption used to secure the pairing process, the lack of hopping during these stages, and the use of Discoverable Mode.

3.3.1 Active Device Discovery: Inquiry Scanning

Because of the lack of a central server, Bluetooth devices need some other way to discover each other existence without interfering with other devices. Such as meachanism is described in the Bluetooth Specification as 'inquiry scanning'. When a master device, such a laptop, is trying to find other devices in the area to connect to, it will broadcast inquiry scan messages over multiple frequencies. Bluetooth slave devices, when they are in a special mode called 'discoverable mode', will respond with a inquiry response message. This message will contain information necessary to initiate the pairing, which is in this case is the Bluetooth address of the slave, and its device clock.

Inquiry Scanning is a very vulnerable stage of the specification. The response messages are sent over a wide range of frequencies and in plaintext, so an attacker would need to know neither the keys nor the piconet attributes to capture the packet and discover the BD_ADDR and the device clock. If a device does not enforce encryption, then attacker can potentially flood all the channels with packets with an approriate header to inject packets into the victim device.

The information sent in this response message is all that is needed to construct a piconet and de-whiten the packets. Even if the information is encrypted, the LAP is sent in plain text, so the Bluetooth packets can still be captured and decrypted later.

3.3.2 Traditional Pairing Process

The Pairing process described above also contains known and well-explored vulnerabilties. Recall that the Pairing process employs two keys:

- Link Key: requires the PIN (usually 4 digits)
- Encryption Key: requires the Link Key, ciphering offset and a random number

The security of the Link Key rests on the PIN being secret. This assumption that the PIN is secret is incredibly broken because even in the case the PIN is not the standard '0000' or '1234', most PIN's are only 4 digits. That results in 10^4 possible PIN combinations to choose from, making brute-force attacks very easy. From there, the data required to generate the encryption key can be easily acquired, breaking the Confidentiality mechanism.

A very well-understood and common Bluetooth attack takes advantage of the vulnerabilities in Discovery and Pairing processes. This attack focuses on forcing the devices to disconnect by flooding the channels with packets indicating the slave has lost the key. This forces the devices to redo the Pairing process, which the attacker can then observe and obtain the link and encryption keys.

The 'Tradtional Pairing Process' has since been succeeded in the Bluetooth Specification v2.1 by a more secure protocol called Secure Simple Pairing. This protocol employs Public Key Encryption instead of the PIN to generate the link key. Nevertheless the vul-

nerability described is still a cause for concern because many devices support the protocols described in the older Bluetooth Specifications for backwards compatability.

3.3.3 Known Attacks

These vulnerbilities allow a range of attacks to be easily carried out. Packet sniffing is a major attack and of primary concern because of the sensitivity of the data that is potentially passed via Bluetooth. Vulnerable Bluetooth keyboards could reveal sensitive information such as passwords and credit card information. Another major attack is packet injection. Knowing the piconet and the BD_ADDR of the victim device goes a long way in allowing an attacker to properly craft packets that will be captured and read by the victim device.

Bluetooth is vulnerable to Man-in-the-middle attacks. In a 2010 paper by Haataja and Toivanen, called 'Two Practical Man-In-The-Middle Attacks on Bluetooth Secure Simple Pairing and Countermeasures', the writers were able to carry out an attack in which they jam all frequencies in order to trick the devices to assume that they have been disconnected. The devices reinitiate the discovery and pairing process. During this phase, the writers successfully use two Bluetooth modules to pose as the master and slave devices, thereby making packet injection and packet authentication attacks possible. Such asn attack is not protected by SSP either, becausethe weakness is not in the encryption mechanism, but in how devices handle disconnections and reconnections. The victim devices are in reality connecting to the attacking devices instead of to each other directly.

Another known attack is Car Whisperer, a project by the Trifinite group that sought to expose the effect of the vulnerabilities described above. The purpose of the project was to make car manufacturers aware of the inherent vulnerabilities in the Bluetooth sets inside cars. This group discovered that not only were Bluetooth sets left in Discoverable mode, but the Pairing Process was also vulnerable because the passkey, which was used in these Bluetoot sets in the place of PIN's, were left at their default values set by the manufacturers. While standing on a bridge over a highway, the researchers were able to use high-powered Bluetooth modules to connect to the Bluetooth modules in cars passing beneath them. Once a connection was established, the researchers were able to inject audio into the car's speakers. They noted that attackers could also potentially eavesdrop on the

car passengers.

Wireless modules are very draining on their hosts. For this reason, manufacturers like to include some sort of 'sleep' mode in their devices when it is not actively transmitting packets. In 2008, a group at the University of Utah demonstrated in the paper "Battery-Draining-Denial-of-Service Attack on Bluetooth Devices", an attack where they repeatedly bombarded a Bluetooth device with malicious requests. This attack not only rendered the device unusable, but heavily drained the battery resources. The only way to defend against such an attack would be to operate the device in so-called 'silent' mode, where the device only listens to the network but under no circumstances responds to any requests. Even operation under 'non-discoverable' mode, would not stop the determined attacker, because the attacker just need to attach the appropriate header (which can be gained via passive listening) to the request mesasges for the victim device to accept the packet.

3.4 Potential Weaknesses

There are other parts of the Bluetooth specification that are intended to reduce interference with other devices, and increase security. However they contain their set of weaknesses, some of which are described below:

3.4.1 Non-Discoverable Mode

To mitigate the attacks described above, Bluetooth devices now usually feature a mode called 'Non-Discoverable' Mode, which default to. This simply means that the device will not attempt to respond to inquiry request messages it sees. Devices will only enter Discoverable mode when a special button is pressed on the device itself. After a timeout that usually last for a few seconds, the device will change back to Non-discoverable mode.

This does not mean the device is safe. As long as the Bluetooth device is powered on it still will accept packets being sent to it with the approprorpriately constructed header (which just contains an error correcting code, a sequence to correlate data, and most importantly, the LAP of the receiving device). This means that this device is still not safe from attacks such as packet injection, as long as the attacker knows the correct LAP for

the device.

3.4.2 Globally Unique BD_ADDR

The BD_ADDR, like the MAC address in your computer, is supposed to be globally unique. This is particularly important in the Bluetooth protocol because it is used by the Bluetooth devices to filter packets not intended for it. However, in 2005, researchers at CSAIL found that this was not really the case. This is potentially a huge security vulnerbility. An attacker could assemble a list of commonly used BD_ADDR's use a device such as spooftooph, which allows the attacker to change the BD_ADDR of his computer's Bluetooth module. He could iterate through a list of commonly used BD_ADDR's until he starts finding packets. Now knowing the BD_ADDR of a device nearby, he can then proceed to launch attacks such as packet sniffing and packet injection.

3.5 What's Tricky?

Unlike the 802.11 wireless module in most laptops, standard BROADCOM Bluetooth modules do not support passive packet sniffing. Recall that any packets being sent to the master device will contain the master's LAP in its header. The master device uses this information to filter out any packets not intended for it before sending it to the CPU. This filtering is carried out on a very low level of the Bluetooth protocol stack.

This makes sniffing Bluetooth very tricky. To sniff any packet sent over a wireless network, your receiver needs to be able to operate in so-called 'promiscuous' mode. In this mode, the receiver receives all the packets it can read without any regard of who it was intended for, and sends it to the CPU. However, implementing 'promiscuous' mode inside Bluetooth, is both unnecessary and expensive from the viewpoint of the manufacturer. Therefore the firmware and the hardware for general-purpose bluetooth modules simply do not support this mode. In the rare case that the hardware can support promiscuous mode, the firmware is usually closed-source. Making the necessary changes to the firmware code to implement 'promiscuous' mode would require extensive reverse engineering of the firmware itself.

3.6 Existing hardware solutions for sniffing Bluetooth

Because standard Bluetooth modules make it incredibly difficult to implement any sort of packet sniffing, you would have to turn to more specialized hardware to do this. Not supporting promiscuous mode is not only a hinderance to security researchers. It is also a hinderance to developers trying to develop Bluetooth related hardware themselves, as debugging is near impossible without some sort of sniffing feature or access to the lower level information contained in the Bluetooth packet. Fortunately, this is enough of a problem that a market for hardware solutions exists for doing just this exists:

3.6.1 FTS4BT

An example of a commercial platform for analyzing this lower-level information in real time is the FTS4BT[7]. It is a hardware and software package created for sniffing and debugging Bluetooth devices. It is an incredibly powerful platform, and the hardware is is specially developed Bluetooth transceiver. Its big brother, the BPA 500, is about the size of a traditional wi-fi router and claims it can support *all adopted specifications, profiles and protocols.*

The FTS4BT would the be the ideal solution to start sniffing packets if it wasn't for the price. Unfortunately the market for these sort of hardware solutions is still a niche market (though this may change as Bluetooth becomes even more popular), so commercial solutions like the FTS4BT tend to be incredibly expensive, starting at a price point of $10,000. This price keeps it firmly inaccessible to many hobbyist researchers.

3.6.2 USRP

A cheaper alternative to the FTS4BT was found. It was the Universal Software Radio Peripheral [6], a hardware platform, which when used with the GNU Radio Project[8], allowed for the user to implement radio projects very easily in software. It has the advantage of having a very large bandwidth, allowed the user to listen to multiple channels simultaneously. Bluetooth security researchers Michael Ossmann and Dominic Spill, consequently developed gr-bluetooth, a collection of tools to do things such as packet sniffing

and following the piconet.

While the USRP is a significant step in the right direction for improving the affordability of hardware solutions for sniffing Bluetooth, at $1000, it is still out of the price point for many hobbyist security researchers. This resulted in a relatively limited community of Bluetooth hackers and consequently, few projects investigating BLuetooth security. This meant the Bluetooth security is actually in a state of 'security by obscurity'. Just because you are not hearing about Bluetooth vulnerabilties all the time does not mean the vulnerbilities do not exist. Joshua Wright, a prominent Bluetooth Security Researcher and co-author of the 'Hacking Wireless Exposed' Series likes to state the following (and thus consequently dubbed by his follwers as *Wright's Law*)

"Security will not get better until tools for practical exploration of the attack surface are made available." -Joshua Wright [28]

3.6.3 Ubertooth One

Michael Ossmann sought to address the problem of the lack of affordable hardware solutions. He saw no reason why tools such as the FTS4BT should be so expensive, though he speculated it might just be due to a lack of competition. He decided to create his own Bluetooth module with three key features:

- Hardware Support for Passive Packet listening
- Open Source Firmware and Software
- Cheap: Price of at most $100

He succeeded, and in 2010 announced his project "Project Ubertooth". The first version of the hardware was dubbed 'Ubertooth Zero'.[21]

Since then, he has released a more powerful Class 1 version of the hardware called "Ubertooth One". While the hardware was released in 2011[25], the firmware and host code still had to be fully developed, but now researchers had a hardware platform they easily access and subsequently build upon. The code for the Ubertooth is still in an very active and relatively premature stage of development, and so cannot be called a complete

Figure 5: Ubertooth One Module [31]

Bluetooth sniffing solution *yet*. A goal of our project was to identify and thus develop code for incomplete areas of this Open Source project.

4 Exploration of tools

In order to examine the current state of Bluetooth security we learned about and employed quite a few different tools, including a three Ubertooth modules, numerous bluetooth devices, and various Open Source software tools for sniffing Bluetooth traffic or exploiting Bluetooth devices. Our study and use of these tools made it clear that more development needed to be done in order to make them good enough for effective Bluetooth vulnerability testing.

4.1 Hardware

Once we settled on using the Ubertooth, we had to see what the current hardware, firmware, and software developed for it were capable of. Installing the tools that were already developed by the Project Ubertooth team we were able to do the following:

- Passively detect Bluetooth devices, even when they were not in discoverable mode
- Detect the Upper Address Part (UAP) of Bluetooth devices (only the LAP is typically sent in communications)
- Use the Ubertooth software to follow a single Bluetooth device as it hopped across channels
- Passively capture Bluetooth traffic on one channel with a Kismet plugin and display captured packets in Wireshark

These features allowed us to learn a lot more about how our Bluetooth devices were communicating and were helpful for general Bluetooth device detection and traffic analysis, but they were not quite sufficient to allow us to really test the possible vulnerabilities in the Bluetooth scheme. We were still unable to capture Bluetooth traffic accross all channels and display it in an easy to analyze format, force a connection with another Bluetooth device, nor inject malicious packets. The next step we took was to test various software packages that might be used in conjunction with the Ubertooth to develop sufficient sniffing or exploit tools.

4.2 Software

We experimented with a number of software tools when analyzing Bluetooth security, including, but not limited to the following:

4.2.1 Ubertooth Host Code

The Ubertooth host code written by the Project Ubertooth team for the Ubertooth module consists of three primary tools: a wireless spectrum analyzer, ubertooth-follow

(code for following a specific Bluetooth device and printing captured packets to the command line), and a Kismet-Ubertooth plugin that captures Bluetooth traffic across one channel and formats and dumps it into a file for further anaylsis in Wireshark [25].

4.2.2 Kismet

Kismet is a wireless network detector and sniffer, which is most commonly used for sniffing 802.11 wireless traffic [17]. However, we mainly used it in conjunction with the Kismet-Ubertooth plugin to detect Bluetooth traffic. One of the best features of Kismet is that it can save all packets captured in a .pcapbtbb file which can be opened in Wireshark and easily analyzed.

4.2.3 Wireshark

We used Wireshark, a popular network protocol analyzer, for displaying the packets we captured with the Kismet-Ubertooth plugin [32]. We have included a sample display Wireshark display of Bluetooth packets captured with our modified version of the Kismet-Ubertooth plugin (Figure 6).

4.2.4 Spooftooph

Spooftooph is a tool developed by JP Dunning, which uses the *bluez* library to allow a user to spoof or clone their BD_Address and device name [30]. We were hoping to use this software to impersonate another device ina piconet and hijack a connection, but unfortunately this was impossible without being able to decode the handshake packets being sent between the current devices in the piconet or having a way to remotely force a single device to disconnect. However, this does remain a very good tool for hiding one's own Bluetooth adapter to prevent potential adversaries from detecting your information or targeting exploits directly at your BD_Address.

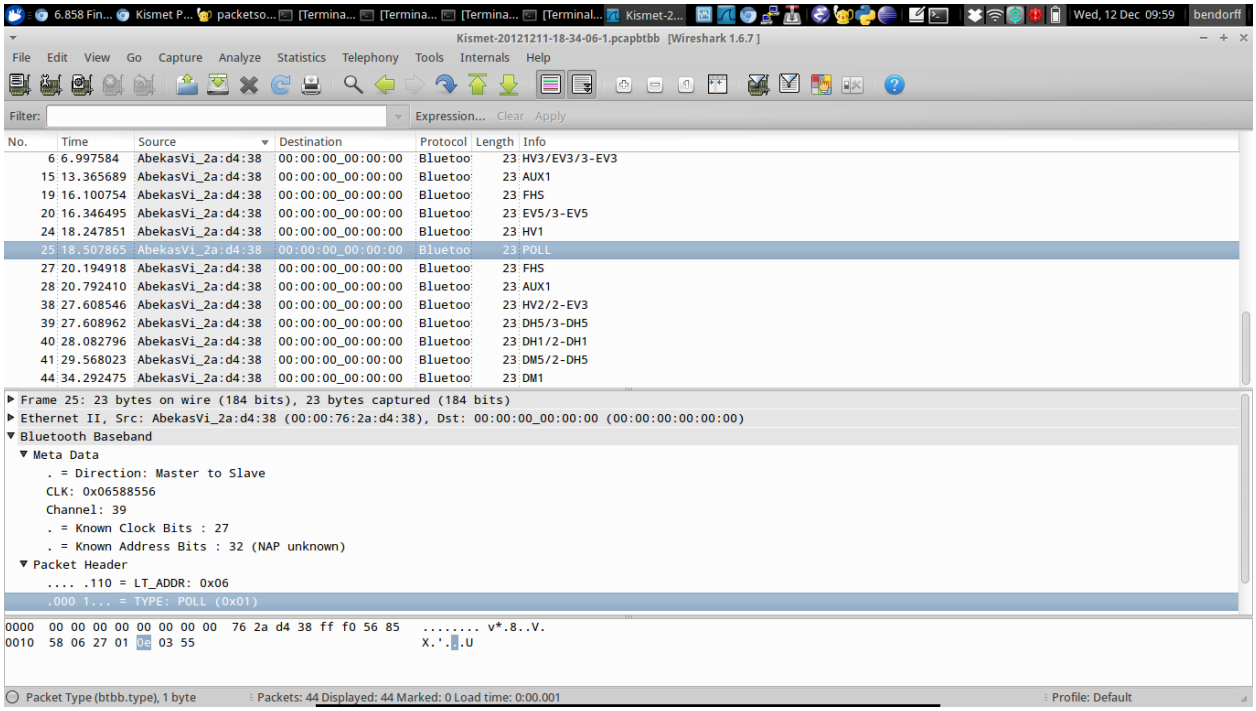

Figure 6: Sample Wireshark Display of Captured Bluetooth Packets.

4.2.5 Pwntooth

Pwntooth is another tool developed by JP Dunning, which is meant to automate Bluetooth security pen-testing [26]. It is a collection of many smaller Bluetooth detection and exploit tools, which can be run together to test the overall security of all Bluetooth devices in the area or a specified Bluetooth device. Unfortunately, we found that the majority of the Blueooth exploits used by the tool would only work when paired and connected with a target device, which right now is only possible when the owner of the targeted Bluetooth device accepts the connection.

4.2.6 HID Attack

The HID Attack, developed by Collin Mulliner, which allows an attacker to inject keystrokes on a victim's computer by pretending to be an HID Bluetooth device, is the best functional Bluetooth exploit that we were able to find [13]. Similarly to Spooftooph,

the HID attack makes use of hciconfig (based on the *bluez* library), to pretend to be a different Bluetooth device, in this case a Bluetooth keyboard that the victim might want to connect to. However, while this exploit was demonstrated to work, it only works for HID hosts in server mode, using unsecure connections. Obviously, this makes it a not very threatening type of attack, though it was interesting to see that it was possible to mimic proprietary devices by using certain BD_Address prefixes.

4.2.7 Tool dependencies

Most of these software tools, the Ubertooth module, and our laptops' internal bluetooth adapters rely on three main libraries, which we will discuss briefly below:

libusb Used by all Ubertooth host software and the Kismet-Ubertooth plugin to communicate with the Ubertooth hardware module.[19]

bluez The official Bluetooth protocol stack for Linux, which is used in various ways by all of the different software tools we looked through and tested [4]. Most of the programs we worked with used the *bdaddress* or *hcitool* packages which are built on top of *bluez*, but some, including the Ubertooth host code and the code we wrote for the Kismet-Ubertooth plugin, use HCI calls directly from the *bluez* library.

libbtbb The Bluetooth baseband library used by Project Ubertooth and gr-bluetooth, which includes all of the code necessary to decode Bluetooth packets[18].

With these tools we were able to locate undiscoverable devices, examine packets intercepted by the Ubertooth module (though it is still hard to interpret their actual content though due to encryption and the fact that slave devices transmit with the master's LAP, not their own), mimic other devices by changing our BD_Address and device name, and inject keystrokes or otherwise exploit Bluetooth devices that we are already connected to. While all of these actions are useful, they are not quite sufficient for really evaluating the security of the Bluetooth protocol and implementation.

Even with these tools, we were still unable to remotely force a Bluetooth device in a piconet to disconnect or to remotely force a connection with another Bluetooth device without that device's owner's input,which rendered the majority of exploits useless. Most of the limitations in the capabilities of these tools were due to the fact that is currently not possible to use these software tools to capture and decode all traffic between two connected Bluetooth devices which are channel hopping together, which is necessary for developing software to undermine or force a connection. For that reason, we decided that we work towards improving current tools for Bluetooth sniffing, so that future researchers and developers will have all of the tools necessary to solve those problems.

5 Implementation

After familiarizing ourselves with the Bluetooth protocol and all of the tools we discussed above, we decided to work with the Kismet-Ubertooth plugin to make Bluetooth sniffing easier and more informative. While the developers of the Ubertooth had already made it possible to follow a Bluetooth device during its channel hopping with their ubertooth-follow tool, their Kismet plugin only scanned for Bluetooth traffic through one channel. Additionally, the packets intercepted by ubertooth-follow are just printed to the command line and cannot be as easily analyzed as those picked up by Kismet, which can be viewed in Wireshark. Therefore, we decided to use the functionality of the Ubertooth and its follow mode to implement channel hopping with a piconet in Kismet, so that future developers in this area will have a better method by which to capture and analyze Bluetooth traffic. Our implementation can be found at https://github.com/cathywu/6858-kismet-ubertooth.

The first major hurdle in implementing this change was to actually understand how all of Project Ubertooth's code worked. After many hours of staring at control flow diagrams and consulting the creators of Ubertooth, Dominic Spill and Michael Ossman, for support, we developed a pretty good understanding of how the hardware, firmware, and software built on top of the Ubertooth module interacts. Fortunately, for us, the firmware already gave us the ability to do the channel hopping we wanted to do and the primary difficulty in

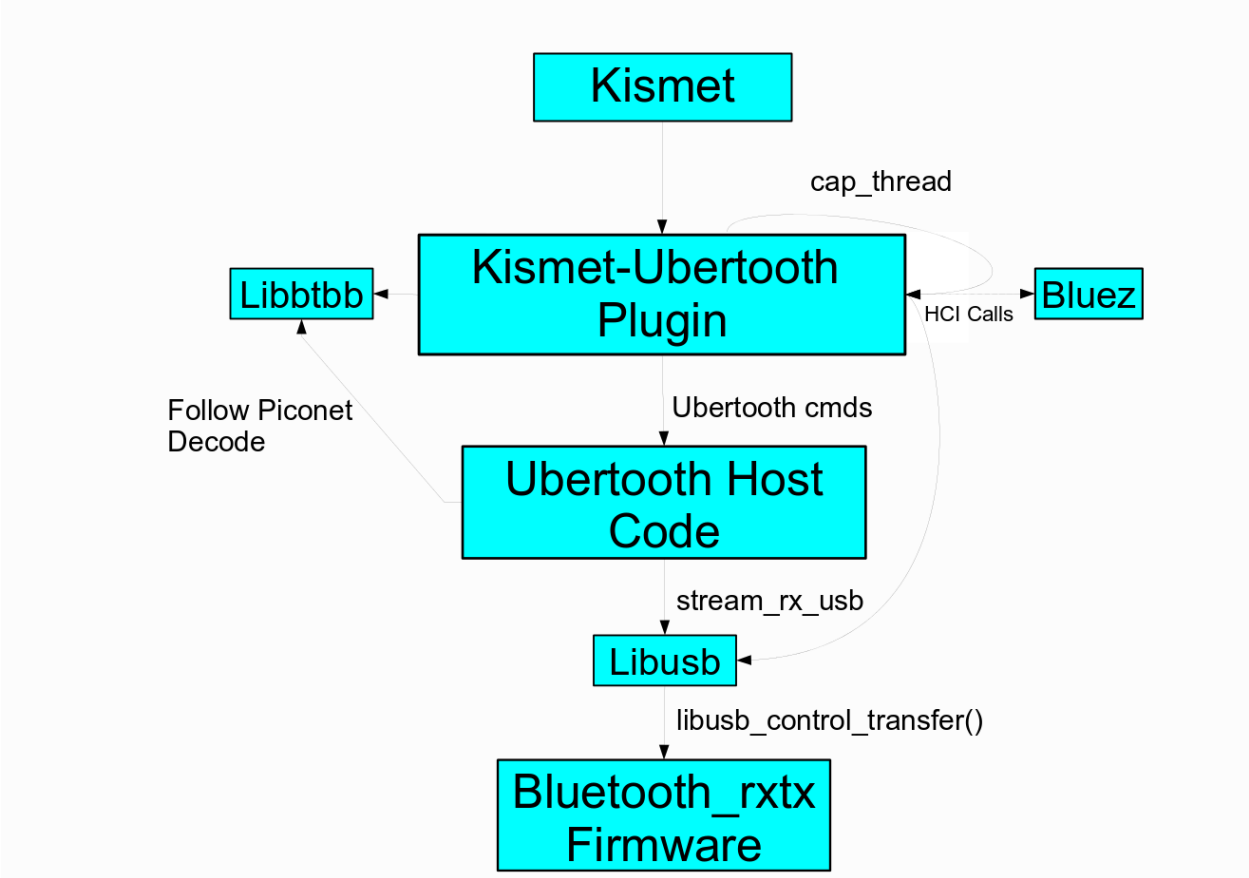

Figure 7: Kismet-Ubertooth Plugin Dependency Diagram.

implementing this new functionality in the Kismet-Ubertooth plugin would be to extend the communication between the Kismet-Ubertooth plugin and the Ubertooth module, so that it could actually follow a given device. The key aspects of this communication and dependency chain are outlined in Figure 7. We implemented this communication by using HCI calls from the bluez library to create a connection with the specified device, which actually presented more problems than we expected, because HCI calls are blocking and will cause the plugin to crash if they are not run in a separate thread from the capture threads.

Using our modified version of the Kismet-Ubertooth plugin, we were able to effectively follow along a piconet consisting of a laptop and a Bluetooth mouse. We were able to pick up many more packets (and many more types of packets) than with the original plugin. Figure 6 shows a sample display of Wireshark from a capture produced by our modified Kismet-Ubertooth plugin.

6 Future Work

We next plan to prepare our modified Kismet-Ubertooth plugin to be merged into the main development branch. More work can be done to make our implementation easy to use. Right now joining a piconet with a specified Bluetooth device requires changing the BD_Address in the code because it is difficult to pass arguments to a Kismet plugin. We are currently finishing up an implementation that allows the user to enter the BD_Address on the command line, storing it in a configuration file to be read by the plugin. An even better alternative that we are working towards is to start the plugin in scan rather than follow mode, to pick up multiple Bluetooth devices, and to allow the user to select the Bluetooth device via the Kismet UI. Once we have made these changes the developers at Project Ubertooth are looking to include our code in the main code base for the Kismet-Ubertooth plugin.

In addition to our contributions, there are many possibilities for future research and development in this area. We encourage future researchers to use our tool and to contribute to Project Ubertooth, to enable a deeper analysis of the Bluetooth protocol and implementation weaknesses. Here are a few suggestions for areas in which more work can be done

in both the Kismet-Ubertooth plugin and in Bluetooth security hacking as a whole:

Implementing AFH for Kismet-Ubertooth plugin The next step in improving our design would be to implement adaptive frequency hopping for the Kismet-Ubertooth plugin, so that it can follow devices with AFH-enabled. Implementing AFH requires determining the AFH map held by the host device, which can be rather difficult. Luckily, the firmware for the Ubertooth module already handles most of the legwork and it would only require a few modifications to add this functionality to the plugin.

Removing dependence on Bluez for piconet hopping A more difficult task in improving the Kismet-Ubertooth plugin would be to remove it's reliance on the bluez library by allowing it to determine all HCI information directly from the Ubertooth module. This is one of the tasks that the Project Ubertooth is looking to tackle at some point and will require quite a bit of low level coding in the firmware and host code of the Ubertooth to implement properly.

Enabling the use of multiple Ubertooth modules Looking further into the future, developing the ability to use multiple Ubertooth modules for following different Bluetooth devices at the same time would be a major step in improving Bluetooth sniffing capabilities. Getting this to work with the Kismet-Ubertooth plugin would require intimate knowledge of the Kismet architecture.

Cracking Bluetooth encryption for decoding packets The long term goal of all of this Bluetooth sniffing is of course the ability to decode and understand all of the packets that are being transmitted. Finding a way to access the link key or encryption key and fully decode traffic may be possible and future researchers should consider whether or not these keys and encryption are actually secure.

Sending "kill" packet to slave device in piconet The simplest Bluetooth exploit would be to remotely cause a slave device to disconnect from the host, ideally without the host knowing about it. This is also the first step in implementing any exploit that takes advantage of a man in the middle attack.

Completely impersonating other Bluetooth device to hijack connection It remains to be seen if this is fully possible, but we believe that future security testers in this field may be able to achieve this level of exploitation against current Bluetooth im-

plementations once the tools for the incremental steps leading up to it have been developed.

7 Conclusion

Bluetooth is an extremely popular short-range wireless technology that still has many security problems. Fortunately, many uses of Bluetooth devices today do not pose huge privacy risks for its users. However, due to its ease of deployment and convenience, an increasing number of serious applications such as keyboards and pacemakers are entering the market; the privacy and potentially the life of Bluetooth users will then rest on the security of Bluetooth. By implmenting piconet following in the Kismet-Ubertooth plugin, we promote effective tools for analyzing Bluetooth security. As history has shown in the development of WiFi security, we believe that the most effective means of communicating the insecurities of Bluetooth is by making Bluetooth sniffing easily accessible to the general public, thereby influencing manufacturers to care about the security of the devices they produce.

References

[1] Blankenbeckler, David. "An Introduction to Bluetooth." The Wireless Developer Network. http://www.wirelessdevnet.com/channels/bluetooth/features/bluetooth.html

[2] "Bluetooth." Another URL. http://www.anotherurl.com/library/bluetooth_research.htm

[3] "Bluetooth Security." Jabra.com. http://www.jabra.com/ /media/Documentation/Whitepapers/WP_Bluetooth_50004_V01_1204.ashx

[4] "Bluez: Official Linux Bluetooth protocol stack." http://www.bluez.org/

[5] Cahce, Wright, and Liu. "Hacking Wireless Exposed, 2nd Edition." Mc-graw Hill, 2010.

[6] "Ettus Research™ USRP." https://www.ettus.com/product

[7] "FTS4BT™ Bluetooth Protocol Analyzer and Packet Sniffer." http://www.fte.com/products/FTS4BT.aspx

[8] "GNU Radio." http://gnuradio.org/redmine/projects/gnuradio/wiki

[9] Gupta, Manoj. "Bluetooth Baseband." Palowireless Wireless Resource Center. http://www.palowireless.com/bluearticles/baseband.asp

[10] Haataja, Keijo and Toivanen, Pekka *Two Practical Man-In-The-Middle Attacks on Bluetooth Secure Simple Pairing and Countermeasures.* WIRELESS COMMUNICATIONS, IEEE Vol 9 No 1 2010 http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05374082.

[11] Harte, L. and Luck, C.J. *Introduction to Bluetooth 2nd Edition; Technology, Market, Operation, Profiles, and Services.* Althos, 2009. http://www.althos.com/tutorial/Bluetooth-tutorial-sleep-modes.html.

[12] Herfurt, Martin "Car Whisperer". 2006.http://trifinite.org/trifinite_stuff_carwhisperer.html.

[13] "HID Attack." http://mulliner.org/bluetooth/hidattack.php

[14] Hodgdon, Charles. "Adaptive Frequency Hopping for Reduced Interference between Bluetooth® and Wireless LAN." Ericsson Technology Licensing, 2003.

http://www.design-reuse.com/articles/5715/adaptive-frequency-hopping-for-reduced-interference-between-bluetooth-and-wireless-lan.html

[15] Huang, Albert and Rudolph, Larry. 2005. "Bluetooth for Programmers" http://people.csail.mit.edu/rudolph/Teaching/Articles/BTBook.pdf

[16] Kostakos, Vassilis. "The privacy implications of Bluetooth." In Proceedings of CoRR. 2008. http://arxiv.org/pdf/0804.3752.pdf.

[17] "Kismet." http://www.kismetwireless.net/

[18] "Libbtbb - Bluetooth baseband library." http://libbtbb.sourceforge.net/

[19] "Libusb." http://www.libusb.org/

[20] Nogales, I. "Bluetooth Security Features." 2006. http://www.urel.feec.vutbr.cz/ra2008/archive/ra2006/abstracts/085.pdf

[21] Ossmann, Michael. "Project Ubertooth: Building a Better Bluetooth Adapter". ShmooCon 2011. http://www.youtube.com/watch?v=KSd_1FE6z4Y

[22] Ossmann, Michael. "Who Owns your Bluetooth?" ShmooCon, 2010. https://www.youtube.com/watch?v=5Y$_9$Nf9MrhY&playnext = 1&list = PL10BC28403A7C1D7C&feature = results_video

[23] Popovski, Petar, Hiroyuki Yomo, and Ramjee Prasad. "Strategies for adaptive frequency hopping in the unlicensed bands." Wireless Communications, IEEE 13.6 (2006): 60-67. http://kom.aau.dk/ petarp/papers/DAFH-AFR.pdf

[24] Premnath, Sriram and Kasera, Sneha. "Battery-Draining-Denial-of-Service Attack on Bluetooth Devices". University of Utah. 2008. http://www.cs.utah.edu/ nandha/Abstract_2008.pdf

[25] Project Ubertooth, Getting Started. http://ubertooth.sourceforge.net/usage/start

[26] "Pwntooth." http://www.hackfromacave.com/pwntooth.html

[27] Spill, Dominic and Bittau, Andrea. "BlueSniff: Eve meets Alice and Bluetooth". USENIX 2007. http://static.usenix.org/event/woot07/tech/full_papers/spill/spill_html/

[28] Spill, Dominic. "Discovering Bluetooth Devices", 2012. http://ubertooth.blogspot.com/2012/10/discovering-bluetooth-devices.html

[29] Spill, Dominic. "Bluetooth Packet Sniffing Using Project Ubertooth".ShmooCon, 2012 https://www.youtube.com/watch?v=HU5qi7wimAM

[30] "SpoofTooph." http://www.hackfromacave.com/projects/spooftooph.html

[31] "Ubertooth One Module." https://www.sparkfun.com/products/10573

[32] "Wireshark." http://www.wireshark.org/