Testing Vulnerabilities in Bluetooth Low Energy

Thomas Willingham
University of Alabama
Tuscaloosa, Alabama
twillingham@cs.ua.edu

Cody Henderson
University of Alabama
Tuscaloosa, Alabama
cthenderson@crimson.ua.edu

Blair Kiel
University of Alabama
Tuscaloosa, Alabama
jbkiel@cs.ua.edu

Md Shariful Haque
University of Alabama
Tuscaloosa, Alabama
mshaque@crimson.ua.edu

Travis Atkison*
University of Alabama
Tuscaloosa, Alabama
atkison@cs.ua.edu

ABSTRACT

Bluetooth Low Energy (BTLE) is pervasive in technology throughout all areas of our lives. In this research effort, experiments are performed to discover vulnerabilities in the Bluetooth protocol and given the right technology determine exploitation. Using a Bluetooth keyboard, practical examples of the Bluetooth Low Energy protocol were able to be provided. Because of the results garnered, it is recommended that Bluetooth Low Energy not be used for any connections that may transmit sensitive data, or with devices that may have access to sensitive networks.

KEYWORDS

Bluetooth Low Energy, Bluetooth Vulnerabilities, Security

ACM Reference Format:
Thomas Willingham, Cody Henderson, Blair Kiel, Md Shariful Haque, and Travis Atkison. 2018. Testing Vulnerabilities in Bluetooth Low Energy. In *ACM SE '18: ACM SE '18: Southeast Conference, March 29–31, 2018, Richmond, KY, USA.* ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3190645.3190693

1 INTRODUCTION

Bluetooth is a wireless protocol that uses radio frequency to transmit data between devices. The technology began development in 1994 and was released into the technology market in 2000 [3, 14]. This new technology provided a solution for users to connect devices and transfer data temporarily. The advantage of Bluetooth over other wireless communication is the ad-hoc nature of its wireless connection that allows for easier data transmission between devices [14]. The Bluetooth technology operates at 2.4 Gigahertz and can reach up to 300 feet in range in its most powerful version [11]. Although the range of Bluetooth frequency is inferior to Wi-Fi, its shorter range is intended to conserve power [11].

*Corresponding author.

ACM SE '18, March 29–31, 2018, Richmond, KY, USA

ACM ISBN 978-1-4503-5696-1/18/03…$15.00
https://doi.org/10.1145/3190645.3190693

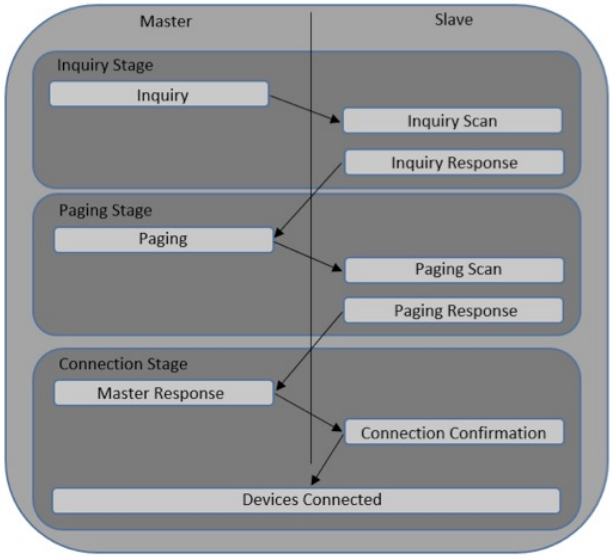

Figure 1: Bluetooth Connection Process

Bluetooth received its name from a Danish King, Harold Bluetooth, who intended to unite Norway's divided groups [3]. The name was chosen because Bluetooth technology seeks to operate as a medium between foreign devices [3]. Version 1.0 and various minor updates were released from 1999 to 2003 [17]. The next innovation occurred in version 2.0 which allowed data to be transmitted at 3 Mbps, and later in version 3.0 at 10 Mbps [17]. The latest update, version 4.0, introduced a standard called Bluetooth Low Energy (BTLE) [17].

The Bluetooth connection process is shown in Figure 1. The first step in order to transmit data via Bluetooth is the Inquiry stage. The Inquiry stage involves the two devices obtaining information about each other on the inquiry channel [9]. One of the devices serves as a master and the other as a slave [9]. The two devices share their BD_ADDR, a unique 48-bit address that consists of its manufacturer ID, its unique device ID, and other device information [9, 16]. The discoverable devices can then decide whether to begin the next stage, paging [9].

The second step of the connection process is the Paging stage. The master device will transmit two ID packets on two of 16 of the possible 32 channels. The master will hop through the 16 channels every 10ms [8]. The slave device listens periodically for a master device, by activating its receiver to listen on one of the 32 possible channels [8]. The slave device will hop to a new channel every 1.28 seconds [8]. The BD_ADDR value determines the channels to use and hopping sequence to perform [8]. Once the slave device receives the paged packet from the master, the slave will then proceed to reply with its own ID packet [4]. The master follows the slave's reply by sending a FHS (Frequency Hopping Sequence) packet back to the slave, who responds yet again with another ID packet [7, 8]. The master reverts to its normal hopping scheme, and sends its first connection stage packet, the POLL packet [4].

To begin the connection stage, the POLL packet merely sends an empty package to the slave; any packet response from the slave is considered acknowledgement of the packet being received by the slave [13]. Upon the acknowledgement from the slave, the devices have an opportunity to create a bonded relationship by pairing [9]. The pairing process involves the two devices storing each other's addresses, names and profiles in memory after authenticating [9]. Although the authentication process varies across different devices, it typically involves the devices matching a PIN code [9]. Once the devices are successfully paired, the bond relationship allows for the two devices to automatically connect whenever they are in range [9]. When a device is in the connection state, it can exist in four modes: Active, Sniff, Hold or Park [4]. A master device in the Active Mode, the standard mode where data can be transmitted and received, attempts to synchronize with the slave on the same channel until it is not addressed then enters one of the three power-saving modes: Sniff Mode, Hold Mode, or Park mode [4]. The Sniff mode allows the slave device to reduce its duty cycle to a set interval and is the least energy efficient mode of the three power-saving modes [4, 9]. The Hold Mode is a mode that the master can command for the slave to enter or the slave can request to enter for a defined amount of time [4]. Lastly, the Park mode is the most energy-efficient mode that the master can force a slave into where the slave ceases to participate in the connection traffic but remains synchronized to the piconet [4, 9].

Outside of the Bluetooth Classic connection standard, the Bluetooth Low Energy standard was released in 2009 [17]. BTLE operates in the same 2.4 GHz frequency as Bluetooth Classic, but has a smaller range than Bluetooth Classic [17]. It uses a Generic Access Profile (GAP) to govern how devices interact with each other [20]. The GAP operates by the master device advertising and scanning with payloads over a set interval [20]. A potential slave responds with its own payload, where it requests a scan response payload from the master device [20]. Outside of the GAP protocol, a master device implementing BTLE can use Generic Attribute Profile (GATT) services to transmit data to multiple slave devices [20]. The GATT services (Figure 2) consist of a series of Services and Characteristics that are transmitted by both the master and slave devices [20]. The transmission object in BTLE consists of Profiles which encapsulate Services, which encapsulate the Characteristics [20]. The Services group related characteristics, which conveniently store an individual data value and a characteristic UUID (16 bit or 128 bit), and are identified by a service UUID (16 bit or 128 bit) [20].

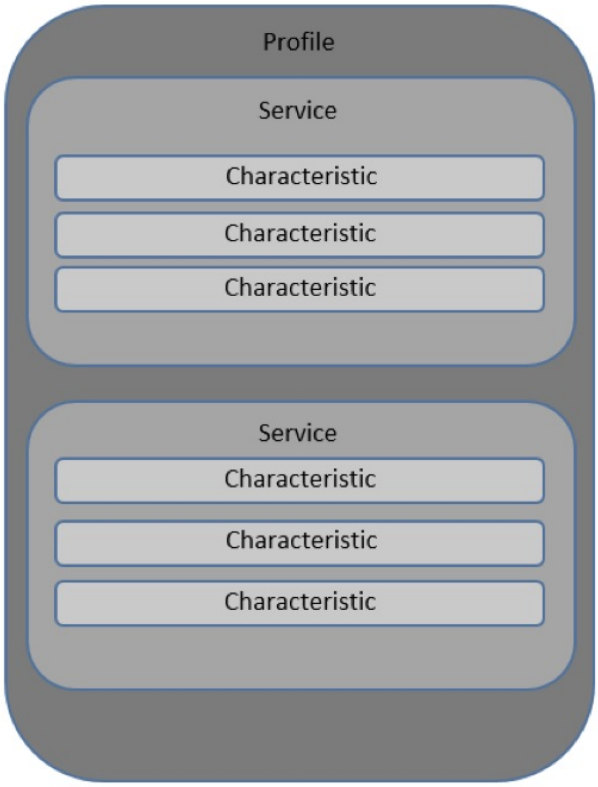

Figure 2: Generic Attribute Protocol (GATT) Object

Upon a connection, the slave device sends a connection interval to the master device for the couple to synchronize for new data periodically [20].

Although the Bluetooth medium is useful for data transmission for the IoT, it comes with many vulnerabilities. A user remaining on the Default Configurations for a device leaves the user vulnerable, especially since one of the most common configurations that comes pre-enabled is Bluetooth inquiry active by default [15]. The device advertising itself on startup is over-looked by users, and leaves the user vulnerable without them knowing it in many cases [15]. Their device info can be used to mimic the actual device's profile [15]. Also, default configurations typically have the lowest security settings configured to allow an easier setup for the user [15].

Another vulnerability of Bluetooth devices is the connection stream, which is vulnerable to eavesdropping and sniffing [15]. To mitigate this risk, Bluetooth attempts to hop frequencies so that the transmission cannot be entirely captured on one frequency [15]. The Bluetooth frequency-hopping-spread-spectrum algorithm decides on which frequency data should be transmitted for the duration that the device specifies [15]. However, if a slave device has been compromised, it has already created a relationship with the master that can then be used to bypass the authentication process, and proceed to transmit data [15]. Exploiting the frequency-hopping algorithm would not be needed in this scenario because the

master and slave have already agreed upon a frequency-hopping sequence [15]. The frequency-hopping algorithm can also be bypassed if the two devices use a weak authentication PIN [15]. The PIN is represented from 8 to 128 bits and brute-force can be used by the attacker until a valid PIN is generated and accepted by the victim device [15]. On the other hand, there are specific programs that listen on all the available frequencies or attempt to imitate the hopping-frequencies after viewing a valid pairing of two devices [15].

Bluetooth is also vulnerable to Man-in-the-Middle attacks like most networks [15]. An attacker can mimic both the master and the slave if the connection process has been observed and the link keys have been obtained; this prevents both devices from communicating directly between each other, and instead the attacker is able to obtain transmitted data before sending data on to the intended recipient [15].

2 RELATED WORKS

This is not the first effort to research exploits on the BTLE protocol. Mike Ryan created the Crackle software that will be used in this paper [19]. Crackle is a software tool that takes advantage of BTLE's unsecured pairing mechanism to find the long-term key (LTK) value needed to decrypt a BTLE conversation. In his paper, Ryan discusses the technical aspects of capturing and decrypting a connection [19].

Das et al. released a paper titled "Uncovering Privacy Leakage in BLE Network Traffic of Wearable Fitness Trackers." [6] In this paper, the authors investigated privacy issues that could be found in the advertising packets of BTLE devices [6]. As an experiment, they used a Bluetooth sniffing device similar to the Ubertooth One to capture a large amount of traffic over a period of time in a local gym [6]. In a controlled setting, they also set up six popular BTLE fitness devices to specifically observe their behavior [6].

One interesting observation they found, especially in their controlled experiments, was the fact that the fitness devices use the same MAC address when advertising [6]. This MAC address can be used over time to track a specific device and potentially a specific individual [6]. They theorize that a malicious person could use these BTLE advertising packets along with video feed from a specific location to attribute a specific device to a specific individual [6]. From that point on, that individual's location could be tracked because of their fitness device's advertising packets [6].

The fitness device's interaction with the user's smartphone causes this vulnerability to exist [6]. To reduce battery usage, the smartphone will often disconnect from the fitness device [6]. It will regularly reconnect to update any information recorded by the device, but there is no need for it to stay connected at all times [6]. While disconnected, the fitness device would switch back to an advertising state [6].

This paper also noted that the BTLE fitness devices did not change their advertising addresses, even between reboots [6]. Randomization of addresses is a feature that Bluetooth can provide, but some manufactures choose not to implement it [6]. Persistent addresses can make it much easier to track the presence of a single BTLE device over time [6].

Other studies have shown that BTLE, like Bluetooth Classic, is vulnerable to a form of denial of service attacks by using conventional microwave ovens [12]. In [12], Louie investigated the correlation between Bluetooth packet losses and the operation of a microwave oven. The study found that the distance between the oven and the Bluetooth devices had a definite correlation with the amount of Bluetooth packets lost [12]. This is due to microwaves using the same frequency as Bluetooth with a much larger power output [12].

The study also mentions the main BTLE pairing vulnerability that we used in our investigations [12]. While this paper does not focus on exploiting the vulnerability, we found it important to note that this is a widely known vulnerability [12].

Albazrqaoe et al. proposed an improved sniffing system based on Ubertooth hardware that could further exploit the BTLE protocol [2]. Their proposed BlueEar system uses different algorithms to more accurately find and follow an established Bluetooth connection between devices [2]. For devices not using any form of encryption, this system could greatly increase the possibility of packet sniffing [2]. In the same paper, the authors also propose slight changes to the BTLE protocol that could prevent their own system from accurately following a Bluetooth connection [2].

3 EXPERIMENTATION AND RESULTS

The goal for this effort was to demonstrate a vulnerability in Bluetooth technology using readily available tools, more specifically, capturing Bluetooth data being transmitted between devices and analyzing that data. Since Bluetooth technology is pervasive throughout our lives in devices such as phones, watches, keyboards, cars, and radios, there is a great need for more in-depth testing of the Bluetooth technology.

3.1 Tools and Setup

To begin the experiment using the Android smartphone and smartwatch, the Android operating system tools that relate to sniffing Bluetooth Traffic were examined. Since Android 4.4, the Android operating system has included a way to sniff the Bluetooth traffic sent and received by the device's Bluetooth radio. Within the Developer Options portion of the Settings app, the "Enable Bluetooth HCI snoop logging" option was enabled. With this option enabled, all Bluetooth packets are saved to a log file within the device's internal storage. Then, interactions on the smartwatch that require it to communicate with the smartphone were executed, and the log file was copied to a computer with Wireshark afterwards.

New versions of Wireshark are able to dissect Bluetooth packets automatically. Wireshark 2.2.0 was used to investigate the log from the smartphone. To help narrow down the packets to look through, 'btl2cap' was used in Wireshark's filter to limit the packets to L2CAP packets that can carry data. The data sent between the smartphone and smartwatch was found to be sent in plain text. As seen in Figure 3, an incoming text message with the message "Yet another message" can be seen within the payload of the L2CAP packet.

This is not the best method of judging the Android platform's security, however. To enable this packet trace, the Developer Options menu and the Bluetooth sniffing option had to be manually enabled by the user. The resulting log is also stored on the device. A

```
6f 64 79 20 48 65 6e 64  65 72 73 6f 6e 4a 1e 0a   ody Hend ersonJ..
0c 6d 65 73 73 61 67 65  5f 74 65 78 74 12 0e 08   .message _text...
02 12 0a 12 08 52 65 73  70 6f 6e 73 65 4a 20 0a   .....Res ponseJ .
11 6d 65 73 73 61 67 65  5f 74 69 6d 65 73 74 61   .message _timesta
6d 70 12 0b 08 05 12 07  28 e3 94 c3 a4 f4 2a 52   mp...... (.....*R
79 08 09 12 75 4a 26 0a  0e 6d 65 73 73 61 67 65   y...uJ&. .message
5f 73 65 6e 64 65 72 12  14 08 02 12 10 12 0e 43   _sender. .......C
6f 64 79 20 48 65 6e 64  65 72 73 6f 6e 4a 29 0a   ody Hend ersonJ).
0c 6d 65 73 73 61 67 65  5f 74 65 78 74 12 19 08   .message _text...
02 12 15 12 13 59 65 74  20 61 6e 6f 74 68 65 72   .....Yet  another
20 6d 65 73 73 61 67 65  4a 20 0a 11 6d 65 73 73    message J ..mess
61 67 65 5f 74 69 6d 65  73 74 61 6d 70 12 0b 08   age_time stamp...
05 12 07 28 b3 c3 c5 a4  f4 2a 0a 15 0a 0b 70 68   ...(.... .*....ph
6f 6e 65 5f 66 6c 61 67  73 12 06 08 06 12 02 30   one_flag s......0
02 0a 12 0a 08 70 72 69  6f 72 69 74 79 12 06 08   .....pri ority...
06 12 02 30 01 0a 26 0a  09 74 65 78 74 5f 68 74   ...0..&. .text_ht
```

Figure 3: Android smartphone's Bluetooth sniffing log

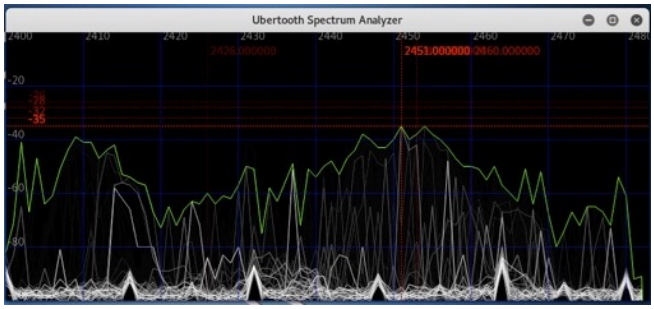

Figure 4: Ubertooth host tools spectrum analyzer

remote attacker would not be able to achieve this without first having physical access to the phone. If physical access was an option, an attacker could access sensitive information much more easily.

To achieve the goal, a device that can capture Bluetooth packets from active connections outside of our control needed to be found. There are some commercial products that would fulfill this requirements, but the cost of the devices made it unfeasible for the scope of this project. Therefore, it was decided that Ubertooth One, a USB 2.4 GHz transmitter and receiver that can read most types of Bluetooth traffic, would be used.

Project Ubertooth is described by its creators as "an open source 2.4 GHz wireless development platform suitable for Bluetooth experimentation." The Ubertooth One is the hardware that can allow a host computer to detect and save packets transmitted by other Bluetooth radios. The corresponding host programs are designed to work on Linux. The 2016.2 release of Kali Linux was used for the host operating system. The Kali distribution comes with certain applications preinstalled that were useful for our project. Wireshark is a protocol analyzer that is commonly used to capture and dissect Ethernet/IP traffic through an interface. With some modification, it can also dissect various Bluetooth protocols. Kismet, another preinstalled tool, is a network sniffer and intrusion detection program. Kismet is used to capture classic Bluetooth packets from the Ubertooth. We acquired the 2015-10-R1 release of the Ubertooth tools and libbtbb from the Ubertooth GitHub repository.

To start, the latest version of the Ubertooth firmware was installed on the device. The instructions for this task can be found within the Ubertooth repository's wiki pages. Updating the Ubertooth firmware was necessary to be more certain that the latest tools would run successfully on the device.

Next, the Ubertooth host tools were installed. The build guide on the Ubertooth repository has a step-by-step guide to extracting, building and installing the necessary packages for the Ubertooth. To run any Bluetooth function on the Ubertooth, the libbtbb and Ubertooth tools steps must be followed. The spectrum analyzer included in the Ubertooth host tools looks similar to that in Figure 4, when the Ubertooth is setup properly. The build guide also has instructions for building and installing Wireshark plugins that can allow Wireshark to read output from the Kismet application. Because of issues when attempting to use Kali's built-in Wireshark plugins, Wireshark 2.0.7 was compiled and installed; the plugins were successfully installed into this version.

Additionally, the Ubertooth plugin for Kismet was installed so that Kismet could begin capturing packets from the Ubertooth hardware. The latest source code for Kismet was downloaded to compile the plugin against. This required us to remove the preinstalled version from the Kali distribution.

Once everything was installed, Ubertooth functionality was confirmed in a number of different ways. The ubertooth-specan-ui command visually shows the 2.4 GHz spectrum and the amount of electronic noise across channels that the Bluetooth channels are allowed to use. The Kismet program was able to interface with the Ubertooth and began capturing classic bluetooth packets. The ubertooth-btle command was able to pipe low energy packets directly to Wireshark so that we could quickly look at the various packets detected by the Ubertooth.

3.2 Deciding on BTLE

After investigating Bluetooth and the complications associated with analyzing it, we decided to narrow our focus to Bluetooth Low Energy. With every passing day, more Bluetooth devices on the market are using BTLE (BTLE, BLE, LE, or Bluetooth Smart) [5]. Since this technology is flooding the market at an ever increasing rate, there is even a greater need for its testing. As discussed earlier, BTLE is a special form of Bluetooth that operates quite differently than other variants of the technology. Using this information and the tools outlined earlier, the goal is to test BTLE to hopefully demonstrate some of its vulnerabilities.

After thorough testing of our Bluetooth devices, we determined which devices have Bluetooth Smart and treat it as a 'first-class citizen'. This narrowed our Bluetooth devices to a FitBit Charge, Logitech Keyboard, and LG G Watch R. The devices used not only had to be able to operate on BTLE, they had to pair on BTLE as well. Part of the decryption and analysis of Bluetooth packets depends on a combination of certain information that is transferred when two BLE devices pair with each other. This is very similar in some ways to cracking a WPA2 secured WiFi connection [1]. These devices are just a few of the many BTLE devices that could have been used.

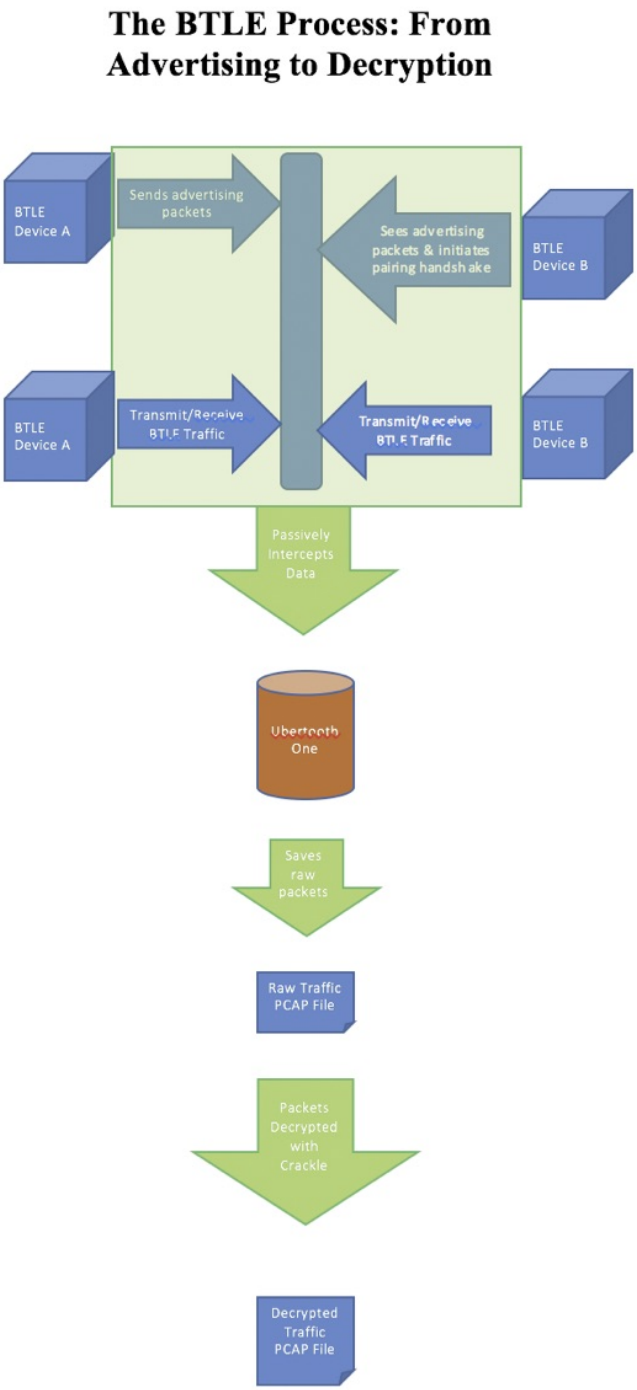

Figure 5: The BTLE Process

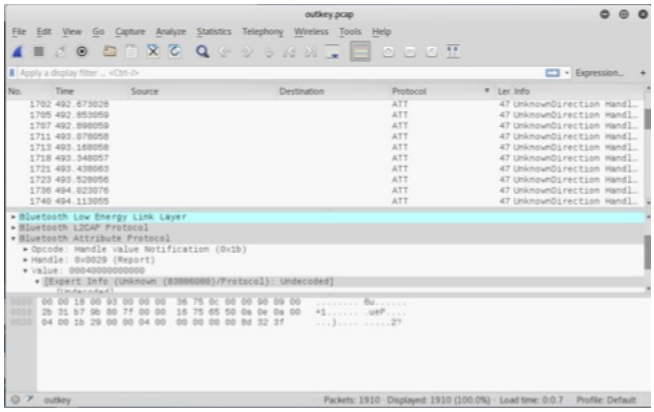

Figure 6: Bluetooth packet capture from the Ubertooth One viewed in Wireshark

3.3 Current Results

After installing the necessary tools, BTLE packets were able to be captured from nearby devices. Unfortunately, the testing location was not the most ideal location at certain times. In the default BTLE follow mode, the Ubertooth will attempt to catch all advertising packets that appear on one of the three advertising channels. At first, a large amount of these advertising packets were captured. The main culprits included Apple TVs that were in nearby rooms. Unplugging these devices and deactivating the Bluetooth antennas on personal computers nearby helped greatly reduce the amount of chatter seen on the advertising channels.

We attempted to use the Ubertooth to capture the connection between the smartphone and smartwatch. When the watch was placed in pairing mode, it sent numerous BTLE advertising packets. We were able to observe these packets and confirm that the MAC address matched that of the smartwatch. However, when the phone initiated a connection with the watch, the Ubertooth would lose all visibility of the connection. We ran this test multiple times to remove the possibility that the Ubertooth was missing the connection request. Because of this, we concluded that the smartwatch was not using Low Energy to actually connect and communicate with the smartphone. Unfortunately, the Ubertooth is not able to read Bluetooth Enhanced Data Rate packets. At the moment, until consumer level Bluetooth sniffing hardware gains the ability to read EDR packets, the Android Wear platform seems to be secure from outside sniffers.

The next target device was a Fitbit Charge, which communicates with the user's smartphone over BTLE. The Ubertooth was set to follow BTLE and log the packets to a PCAP file which could then be read on Wireshark. We repeatedly attempted to pair with the Fitbit with a Samsung smartphone running Android 7.0. After repeated attempts, the Ubertooth was finally able to catch the beginning of the connection and start following the conversation. Despite capturing some traffic between the Fitbit and the phone, we were not able to interpret any of the packets, mainly because we did not know what format or content the data transferred between the devices would take.

As we continued to refine our process, we decided a BTLE enabled keyboard would be a better candidate target as we would hopefully be able to find a correlation between specific keys and the packets sent to the host. In our testing, a Logitech K780 was used and alternated between a Samsung smartphone running Android 7.0 and a Mac Mini running Mac OS Sierra as the host. Again, we repeatedly paired the keyboard to a host device in an attempt to capture the pairing request with the Ubertooth and begin following the connection. We did capture a number of traces, but we encountered problems when attempting to use crackle to decrypt the stream. Despite the Ubertooth attempting to follow the connection, key packets used to decide on the LTK for the connection were not captured. This could have been caused by the noisy environment we were running in at the time. Eventually, we did successfully capture a complete pairing and subsequent keystrokes. Crackle was able to find the LTK and decrypt some of the packets within the connection. We found that BTLE keyboards, like traditional keyboards, do not send a plaintext letter to the host. Instead, we were able to find the corresponding codes for the keys we repeatedly pressed while the Ubertooth was recording the connection. With that, we can successfully show that we can capture and decode keystrokes from a BTLE keyboard connection when we observe the initial pairing of the keyboard with a host device.

For our specific purposes, we are targeting a vulnerability in the Advertising packets that are transmitted over three channels between the Slave device and the Master device. The Ubertooth device attempts to listen to the entire connection process between the Slave device and the Master device. When using an Ubertooth device, one Ubertooth can only listen to one of these channels at a time. This is sometimes problematic for trying to capture the entire connection process, which is necessary for capturing the PCAP data files that are needed. Upon starting the Ubertooth device, it will attempt to listen to one of those advertising channels. If the Ubertooth is able to detect a connection request and response, it is likely that the device can calculate the channel hopping sequence, follow the connection, and save the packets as PCAP files. If all conditions to this point are successful, the user can output the PCAP files to a destination folder.

At this point, even though we may have PCAP files of the traffic between the two devices, it will most likely be encrypted. We still must find a way to view the encrypted data. In order to have a chance of unencrypting the data, the encryption request and response must be present in the PCAP file. A vulnerability in the BTLE protocol allows cracking of the LTK that is used by both devices to encrypt and unencrypt the data. Given the short length of the key, it can be brute-forced fairly easily. Crackle is one such tool that can brute-force the key. It is noted that the LTK is reused between the Slave device and Master device for future BTLE transmission [10]. If the user is able to crack the LTK with Crackle, future transmissions could also be decrypted between the two devices [18]. With the LTK now cracked, captured PCAP data can be decrypted with Crackle and sent to a destination folder [18]. With the decrypted data in our possession, we can open the decrypted data in Wireshark and analyze the BTLE data that was sent between the two devices.

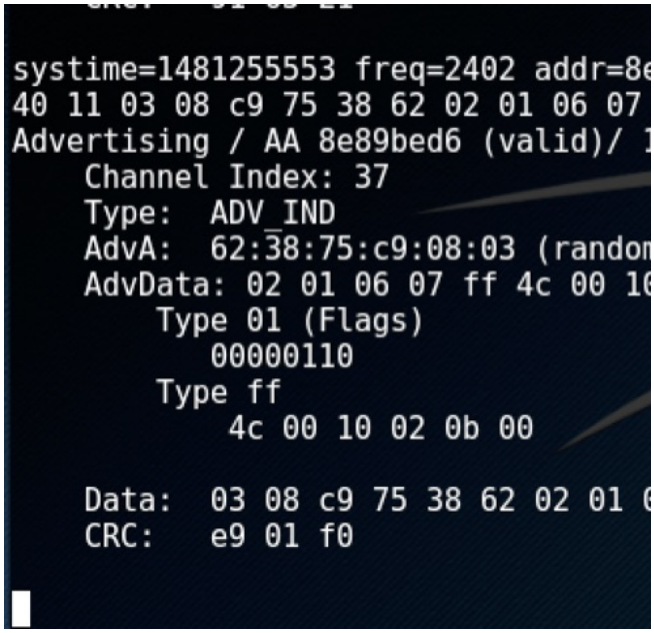

Figure 7: Monitoring BTLE Packets on Channel 37 with the Ubertooth One

4 FUTURE DIRECTIONS

Moving forward with this effort, we want to build on our current success. After having watched the keyboard pair, crackle can find the LTK used for the connection. Since the LTK can be used for future traffic, it would be useful to test the ability of the Ubertooth to follow the connection and capture packets after the original capture, then attempt to decrypt the new traffic with the same LTK. Other research also needs to be done using the Ubertooth's promiscuous mode to find the keyboard's already established connection. This would increase the damage that could be done via the BTLE protocol significantly. It is worth noting that, in our experience, the Ubertooth would often freeze after a few seconds of listening in promiscuous mode.

We also want to attempt the same steps with a smartwatch. We currently have an LG G Watch R at our disposal that can be connected to our Samsung Android smartphone. After examining the application-level Bluetooth packet traces recorded by the Android operating system, we know that the smartwatch apps communicate with the smartphone apps at least partially with plaintext. We want to attempt to find the same plaintext by capturing the link-level packets with the Ubertooth. This will be more time consuming due to the time it takes to pair and reset the smartwatch itself as compared to the BTLE keyboard. Since the Smartwatch transmits data at times that may not be triggered by the user, it may be possible to externally trigger a transfer of data to or from the watch that could be intercepted.

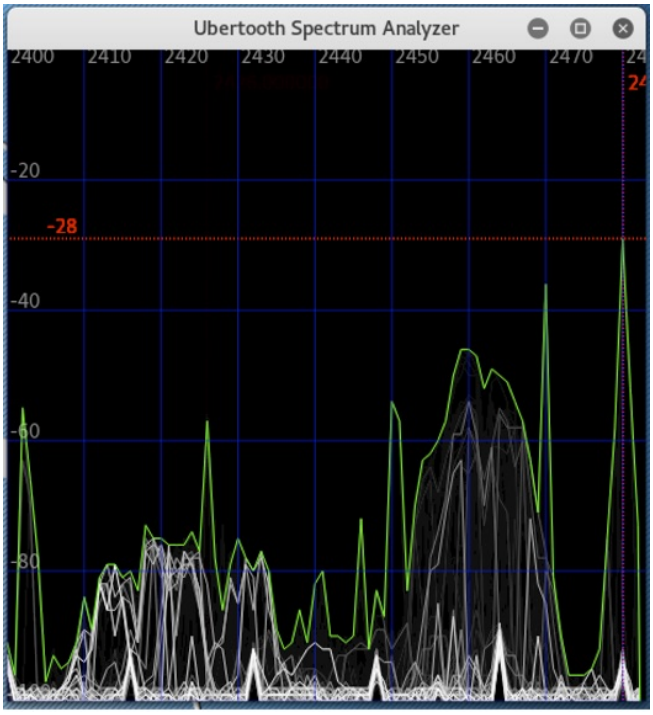

Figure 8: 2.4 GHz Spectrum Analysis with the Ubertooth One

5 CONCLUSION

Today's market is flooded with devices that support BTLE - from lightbulbs to smartphones, the technology is everywhere. Throughout our research we have found that there are known vulnerabilities in the Bluetooth protocol, and that given the right technology, those vulnerabilities can be exploited. There are many practical examples of the possible exploits of BTLE, but this paper just highlights one, Bluetooth keyboards. Since we transmit most everything that we input into our computers through a keyboard, it is crucial that we can trust the integrity and security of the keyboards that we use. Unfortunately, BTLE cannot provide that assurance in it's current state. Based upon this, we recommend that BTLE not be used for any connections that may transmit sensitive data, or with devices that may have access to sensitive networks.

REFERENCES

[1] Aircrack. 2016. Tutorial: How to Crack WPA/WPA2. (2016). https://www.aircrack-ng.org/doku.php?id=cracking_wpa
[2] Wahhab Albazrqaoe, Jun Huang, and Guoliang Xing. 2016. Practical Bluetooth Traffic Sniffing: Systems and Privacy Implications. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 333–345.
[3] Bluetooth. 2016. The story behind Bluetooth technology. (2016). https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth
[4] M. Bundalo. 2016. Bluetooth Baseband. (2016). http://ecee.colorado.edu/~ecen4242/marko/Bluetooth/Bluetooth/SPECIFICATION/Baseband.htm
[5] J. Coombs. 2016. The straight goods on Bluetooth: How many consumers have it on? (2016). http://blog.roverlabs.co/post/117195525589/the-straight-goods-on-bluetooth-how-many
[6] Aveek K Das, Parth H Pathak, Chen-Nee Chuah, and Prasant Mohapatra. 2016. Uncovering privacy leakage in ble network traffic of wearable fitness trackers. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*. ACM, 99–104.
[7] P. Dziwior. 2005. ACL (asynchronous connectionless) links. (2005). http://www.dziwior.org/Bluetooth/ACL.html#FHS
[8] P. Dziwior. 2005. Paging. (2005). http://www.dziwior.org/Bluetooth/Paging.html
[9] H. Ford. 2016. GATT. (2016). https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt
[10] Ins1gn1a. 2015. Introduction into Bluetooth and Bluetooth Low Energy Testing. (2015). https://www.ins1gn1a.com/introduction-into-bluetooth-and-bluetooth-low-energy-testing/
[11] J. Jacobs. 2016. What frequency is Bluetooth? (2016). https://www.techwalla.com/articles/what-frequency-is-bluetooth
[12] A. Louie. 2016. Information Leakage in Mobile Health Sensors and Applications. (2016). https://thawproject.files.wordpress.com/2014/07/anthony-louie-final-information-leakage-in-mobile-health-sensors-and-applications.pdf
[13] S. Mahmud. 2005. Bluetooth Technology. (2005). http://ece.eng.wayne.edu/~smahmud/PersonalData/PubPapers/Handout_Jul12_05.pdf
[14] D. Nield. 2016. What is Bluetooth? (2016). http://www.techradar.com/how-to/computing/what-is-bluetooth-1323284
[15] Tu C Niem. 2002. Bluetooth and its inherent security issues. *Global Information Assurance Certification (GIAC) Security Essentials Certification (GSEC), Research Project, Version 1.4 b* (2002).
[16] I. Poole. 2016. Bluetooth network connection and pairing. (2016). http://www.radio-electronics.com/info/wireless/bluetooth/networks-networking-connections-pairing.php
[17] I. Poole. 2016. Bluetooth Technology Tutorial. (2016). http://www.radio-electronics.com/info/wireless/bluetooth/bluetooth_overview.php
[18] M Ryan. 2014. BLE Fun With Ubertooth: Sniffing Bluetooth Smart and Cracking its Crypto. (2014). https://blog.lacklustre.net/posts/BLE_Fun_With_Ubertooth:_Sniffing_Bluetooth_Smart_and_Cracking_Its_Crypto/
[19] Mike Ryan et al. 2013. Bluetooth: With Low Energy Comes Low Security.. In *WOOT*.
[20] K. Townsend. 2016. Introduction to Bluetooth Low Energy. (2016). https://learn.adafruit.com/introduction-to-bluetooth-low-energy