Investigating Wii Remote Communication

Luke Janik
Northeastern University
Boston, MA
janik.l@northeastern.edu

Corrine Cella
Northeastern University
Boston, MA
cella.c@northeastern.edu

Abstract

For this project, we investigate the communication between the Nintendo Wii Remote Controller and the Wii Console. The devices use classic Bluetooth technology to coordinate functionality such as the rumble feature, speakers, and more. We evaluate the weaknesses of these communications and exploit them to spoof messages that allow us to control the Wiimote's physical responses. We use the Ubertooth One to eavesdrop on the communication and propose a theoretical approach to a man in the middle attack between the devices.

---

1 Introduction

For this past semester, we have been looking into how a Wii and Wiimote communicate wirelessly. The Wii video game console was released by Nintendo in 2006, and the Wii Remote Controller (Wiimote) is how users can interact with the Wii and play video games. We know that the Wii and Wiimote communicate over Bluetooth, so our goal for this project was to learn how Bluetooth works and observe the traffic between the two devices to see how vulnerable the communication is. Our ultimate goal was to learn how the communications were structured in order to create our own interactions with the Wii and Wiimote using an external Bluetooth transmitter and spoofed messages to activate specific features, such as the speaker in the Wiimote and the rumble feature.

Unfortunately, due to the spread of Covid-19, the closure of campus and social distancing, our ability to complete the project as initially intended was impacted.. After rescoping our project, we decided our new deliverable would be to observe the traffic between the Wii and Wiimote, evaluate the potential of overshadowing the connection, and modifying messages from a conceptual perspective rather than an experimental one.

After conducting thorough research, we concluded that a man in the middle attack can be performed between the Wii and Wii Remote by spoofing the connections between the devices with a single Bluetooth enabled laptop and an Ubertooth device and additional software.

We will first offer a short background of the Wii Remote and discuss the communication protocols that are used by the device in section 2. Next we will review the security of Bluetooth and the associated protocols and how we can take advantage of the known weaknesses and implement them experimentally and theoretically in section 3. We will conclude the paper with a summary of our results, a discussion of the obstacles we encountered, offer some ideas for future work, and note our takeaways from this project.

2 Background

2.1 Wiimote and Bluetooth

The Wiimote refers to the remote used to interact with a Wii, a video game console. It was very popular in the mid 2000s when it first came out, but has since been discontinued. The Wiimote and Wii communicate with Bluetooth to handle and coordinate the functionality each provides to the user. This communication begins with the pairing procedure, indicated to the user by blinking blue LED lights at the bottom of the Wiimote. By pressing the 1 and 2 buttons, or the "sync" button, the Bluetooth connection will attempt to be established by first sending the "Require Authentication" message [10]. This step in the procedure puts the Wiimote into discoverable mode. This PIN is then verified and will either be the Bluetooth address of the Wii or Wiimote backwards, depending on which buttons were pressed to start the pairing procedure [10]. Once this connection is acknowledged and accepted with the "Authentication Accepted" message, the process can continue with the Human Interface Device (HID) connection establishment [10].

2.2 HID Connection

I/O	ID(s)	Size	Function
O	0x10	1	Rumble
O	0x11	1	Player LEDs
O	0x12	2	Data Reporting mode
O	0x13	1	IR Camera Enable
O	0x14	1	Speaker Enable
O	0x15	1	Status Information Request
O	0x16	21	Write Memory and Registers
O	0x17	6	Read Memory and Registers
O	0x18	21	Speaker Data
O	0x19	1	Speaker Mute
O	0x1a	1	IR Camera Enable 2
I	0x20	6	Status Information
I	0x21	21	Read Memory and Registers Data
I	0x22	4	Acknowledge output report, return function result
I	0x30-0x3f	2-21	Data reports

Figure 2.2: Table showing the message information communicated between the Wii and Wiimote. [10]

The HID connection uses Bluetooth L2CAP protocols to handle its messages [10]. There are two pipes where messages can travel, a control pipe and a data pipe, however the Wii and Wiimote primarily use the data pipe for all of their communications [10]. The two main details required in each message are an indicator of whether it's an input or output message and the ID which represents the functionality that is being addressed [10]. This can be seen in Figure 2.1.

2.3 Bluetooth Security

Bluetooth devices in communication with one another form an ad-hoc network called a piconet. The device that initiates the connection is called the master and the other device is the slave. The master and slave go through a pairing process that has multiple weaknesses. First, the Bluetooth protocol relies on the Bluetooth Device Address (BD_ADDR) which is used to identify the device. The address is used throughout the protocol for communication, identity and authentication.

Figure 2.3 shows the layout of the address. In all Bluetooth devices including the Wii Remote, the Nonsignificant Address Part (NAP) is the first two bytes and is assigned to the manufacturer and is publicly available. The last three bytes are assigned by the manufacturer and are unique to the device (in this case, the Wii Remote). Because the BD_ADDR is used to create the shared link key for communication, it should be kept secret. When the Wii Remote is in discoverable mode, however, the BD_ADDR is advertised to the host. Furthermore, the LAP is sent in plaintext during communication with legitimate devices and can be monitored easily [2].

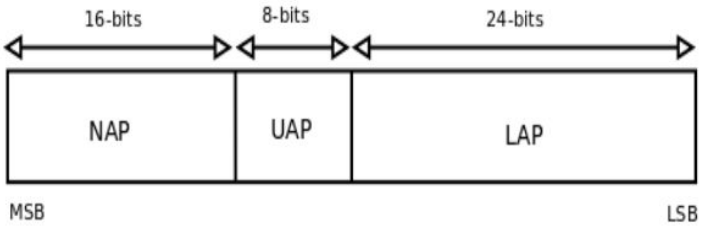

Figure 2.3: Bluetooth Device Address format. [7]

The Bluetooth protocol makes the assumption that once a connection is established between the master and the slave that it will remain secure with the keys it generated. The protocol also relies on the short range of Bluetooth waves which restricts adversaries to a close proximity to the legitimate devices. We will take advantage of these assumptions to perform eavesdropping on the Wii Remote and offer theoretical approaches to spoofing messages and an implementation of a man in the middle attack between a Wii Remote and a Wii Console using just a laptop as the attacker.

In order to do the attack we will need to perform passive eavesdropping to learn the LAP of the Wii Remote and the address of the Wii Console. Next, we must authenticate the laptop with the Wii Remote to spoof the connection with the Wii Console. To perform the man in the middle attack, we also need to use the laptop to spoof the Wii Remote to the console. Lastly, we need to understand the message format to replay or recreate the commands sent between the legitimate

devices. The following sections explain the steps we took to understand this complex communication methodology and the theoretical approaches we would take to complete the attack.

3 Implementation

3.1 Device Authentication

The Wii Remote uses standard Bluetooth technology to connect with hosts. Because Macbooks, like most laptops, have Bluetooth chipsets, we attempted to connect the WiiMote to a computer. We put the WiiMote in discoverable mode by pressing the red button in the battery case of the device and opened the Bluetooth Preferences on a Macbook. We could see a new device in the discoverable devices list and by the name of it, we can assume it is the Wii Remote. The Bluetooth address of the device is revealed by right clicking the device in the preferences as shown in Figure 3.1. Upon hitting connect, a new window appears asking for a PIN. We found in our research that this could be the Bluetooth address of the Wii Remote backwards [10], but we were unsuccessful in cracking the PIN after numerous attempts.

Figure 3.1: Connecting to the WiiMote from a Macbook Pro with Bluetooth.

The Wii Remote enters discoverable mode by pressing the 1 and 2 buttons simultaneously or pressing the red button inside the battery casing of the remote. This process must be done manually the first time the Wii Remote connects to a Wii Console. While testing the manual functionality of the devices, we noticed the Wii Remote loses connection with the Wii Console somewhat frequently (at least once per hour) which requires the user to put the device in discoverable mode again to connect to the console. Furthermore, when the Wii Console is turned on, the Wii Remote is discoverable for a short time period (about 5 seconds) while it reauthenticates with the Wii Console. During this time, we can passively watch the device from the Bluetooth Preferences menu on the MacBook and know when the device is in discoverable mode. We can use this time to authenticate ourselves with the Wii Remote to spoof the Wii Console.

While we were not able to guess the PIN to establish the Bluetooth authentication, many sources indicate that the PIN is either four or six characters long. The total number of possible PINs is between 10^4 and 10^6. The PIN could be guessed easily using PIN cracking programs like [9]. For the remainder of this attack we will assume that this program works properly, that we have correctly guessed the PIN and can authenticate with the Wii Remote if it is in discoverable mode.

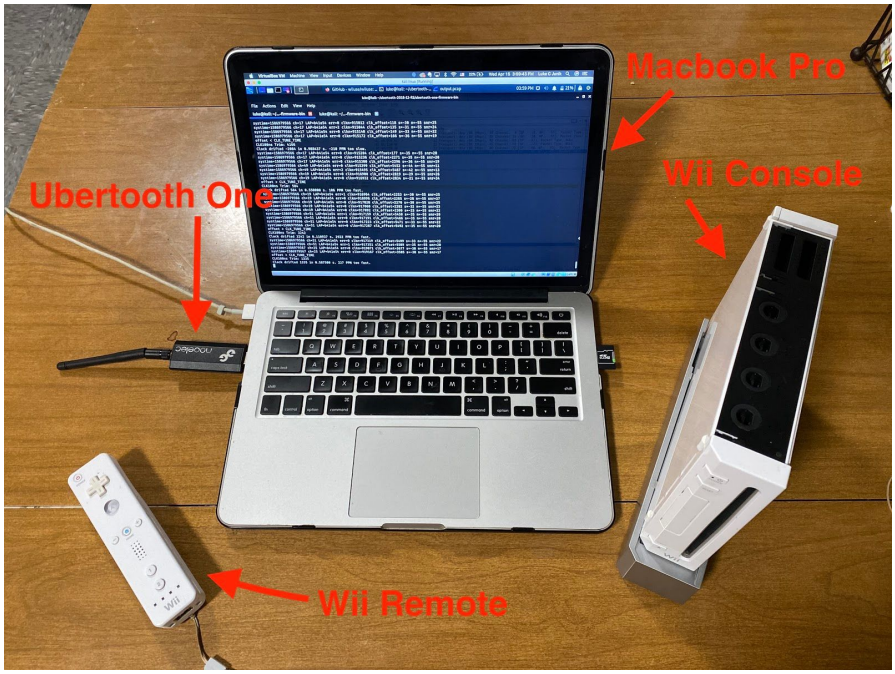

Figure 3.2: Hardware setup with the Ubertooth One.

3.2 Passive and Active Eavesdropping

To understand the communication between the Wii Remote and the Wii Console we must sniff the Bluetooth traffic between the devices. We used the Ubertooth One device for this purpose.

Ubertooth One

The Ubertooth One is an open source project created by the company Great Scott Gadgets. It is a wireless development platform that includes both hardware and software to allow users to experiment with Bluetooth technologies. The device is capable of sniffing both Bluetooth Low Energy (BLE) and Basic Rate (BR/EDR) Bluetooth Classic connections.

The original hardware release was the Ubertooth Zero in 2010, and the current model is the Ubertooth One which was released in 2011. We ordered the Ubertooth One, which retails for around $120 USD. The hardware includes an RF connector for an antenna, a CC2400 wireless receiver, and a microcontroller with USB 2.0 nad is capable of transmitting and receiving wireless signals at 2.4 GHz. The accompanying software for the device can be found on Great Scott Gadget's github page [4]. We followed the setup procedure in [1] to get the Ubertooth up and running on a Kali Linux virtual machine.

We started by putting the Ubertooth in survey mode as a form of passive eavesdropping using the command `$ ubertooth-rx -z`. This mode configures the Ubertooth to listen to and report on all Bluetooth signals that it observes [5]. We ran this command for two 5 minute segments. We first ran the program without connecting the Wii Remote to any host, and simply put it in discoverable mode. Next, we sniffed the traffic while connecting the Wii Remote to the Wii and used the Wii Remote with normal behavior as a regular user.

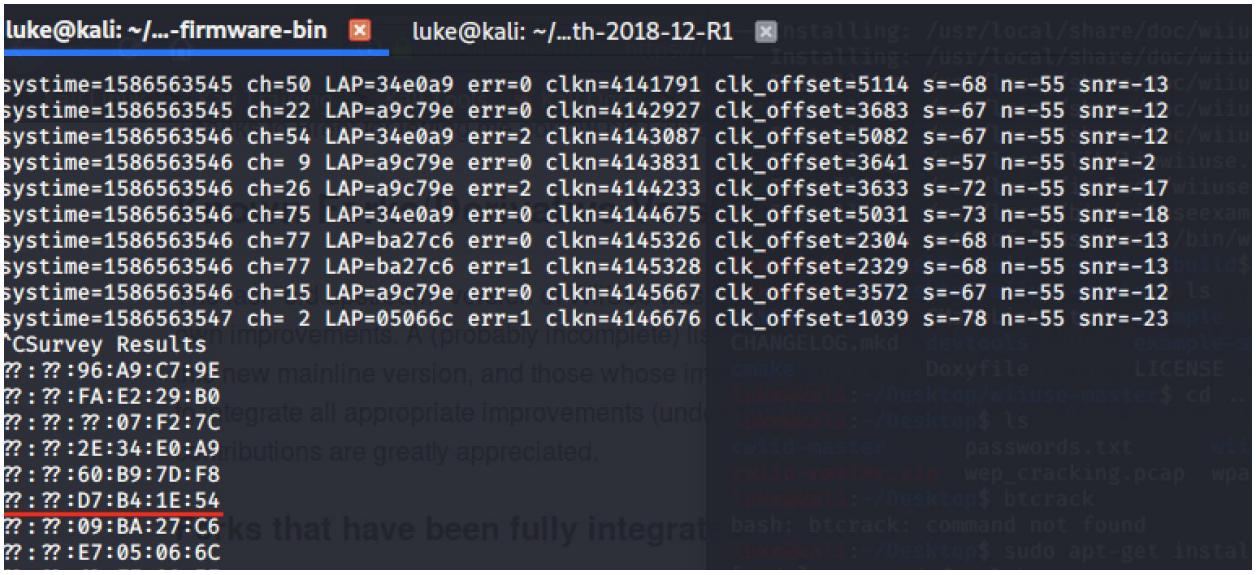

Figure 3.2.1: The console output of Ubertooth's survey mode underlining the address of the Wii Remote.

As shown in Figure 3.2.1, this command outputs the system time, the Bluetooth channel the signal was received on, the LAP of the Bluetooth address, the Bluetooth clock, the clock offset, the received signal strength, the noise level, and the signal to noise ratio. When the program is finished (after a set time or when the user manually ends it) the LAPs of all discovered devices are printed. The sixth address in Figure 3.2.1 shows that LAP for the Wii Remote, indicating that some of its packets were captured. We ran the program again, this time only capturing packets from the Wii Remote and ignoring all other packets using the flag `-l b41e54` to specify the LAP of the Wii Remote as a form of active eavesdropping.

```
luke@kali:~/ubertooth-2018-12-R1/ubertooth-one-firmware-bin$ ubertooth-rx -l b41e54 -u d7 -q output.pcap
systime=1586823603 ch=45 LAP=b41e54 err=0 clkn=9449 clk_offset=4962 s=-38 n=-55 snr=17
offset > CLK_TUNE_TIME
CLK100ns Trim: 2712
systime=1586823605 ch=56 LAP=b41e54 err=0 clkn=18026 clk_offset=3020 s=-36 n=-55 snr=19
systime=1586823605 ch=73 LAP=b41e54 err=2 clkn=18324 clk_offset=3022 s=-81 n=-55 snr=-26
systime=1586823606 ch=47 LAP=b41e54 err=0 clkn=20127 clk_offset=4510 s=-37 n=-55 snr=18
systime=1586823607 ch=70 LAP=b41e54 err=1 clkn=22424 clk_offset=187 s=-81 n=-55 snr=-26
offset > CLK_TUNE_TIME
CLK100ns Trim: -2063
```

Figure 3.2.2: Console output from capturing packets only from the Wii Remote.

From the output in Figure 3.2.2 it is clear that each packet is being received on a different channel as expected. Bluetooth transmits data in packets on 79 channels and hops between the channels at a rate of 1600 hops per second. One limitation of the Ubertooth is that it cannot collect every packet because it cannot listen to all the channels simultaneously, so it cannot be determined if there are missing packets in our collection but our assumption is that some packets were lost.

3.3 Message spoofing

We saved the packet capture to a PCAP file and investigated it further in Wireshark. We noticed that not all packets have data, and the ones that do are encrypted. The next step would be to try to decrypt the data in the packets. Theoretically to do this, the link key between the WiiMote and the authenticated device could be derived from the PIN and hardware address of the Wiimote as mentioned earlier.

Once the device is authenticated, we need to send messages to the Wii Remote as if we were the Wii Console. The types of messages that the Wii Remote receives are shown in Figure 2.1, and some examples include making the Wii Remote rumble or play noise. We acknowledge two types of spoofing: replaying and recreating messages. Replaying messages would involve taking the packets that were captured from the Wii Console in a previous connection and sending them again and modifying fields like the timestamp and clock. However we would like to have more control over the messages we send, and therefore we chose to recreate the messages. Because of the previous research completed on the Wii Remote status and data reporting, we have most of the knowledge required to reconstruct the data of the message [10]. Next, we would use the link key that is given during authentication to encrypt the data, and use the packet headers that were found while eavesdropping the communication from Wireshark. An example of a message that would turn on an LED light may look like Figure 3.3 [11].

```
unsigned char buf[3];
buf[0] = 0x52; // data output
buf[1] = 0x11; // LED reportID
buf[2] = 0x10; // player 1 LED

send(out_socket, &buf, 3, 0); // out_socket = outgoing bluetooth connection socket
```

Figure 3.3: Example message construction for turning on a Wii Remote's first LED light. [11]

3.4 Man in the Middle Attack

To perform the MITM attack, we must spoof both ends of the communication. Most importantly, we need to trick the Wii Remote into thinking it is communicating to the Wii Console. We would implement this by waiting for the Wii Remote to enter discoverable mode, which as mentioned earlier, occurs when the Wii Console is turned on. During this time, we will spoof the hardware address of the Wii Console on the laptop, and accept the connection to the Wii Remote as it probes for the Wii Console. This would be successful because the Wii Remote can only connect to one device at a time, so whichever device responds faster will establish the connection. This requires the attacker to be physically closer to the Wii Remote. However, it is unclear if a new link key needs to be recreated each time the Wii Remote enters discoverable mode, and if it does not, then we could save some time by reusing an older link key if it exists.

Simultaneously, we will spoof the BD_ADDR of the Wii Remote and send the requesting probe to the Wii Console, tricking the Wii Console into thinking it is communicating to a Wii Remote using the same methodology as above. Because reliable transmission is not guaranteed in

Bluetooth [3], we have no reason to believe that the additional round trip time of communication between the Wii Console and the Wii Remote will have any effect on the attack. Furthermore, most actions between the Wii Remote and the Wii Console do not require a response from the other device. For example, if the message from the Wii Console to the Wii Remote to rumble is lost, it has no reason to resend it. Figure 3.4 shows the MITM attack setup.

Figure 3.4: Man in the Middle Attack setup

4 Discussion

In this section we will discuss the obstacles we faced, some areas for future work, and the key takeaways that we learned during this project.

4.1 Challenges

Our first challenge appeared when we attempted to connect the Wiimote to a computer. This was our initial attempt to observe the traffic that a Wiimote would produce. During our research, we found programs that would have allowed us to see this traffic when connected to a computer. However, as mentioned before, technology has evolved a lot since the Wii and Wiimote were popular. The programs we found did not work for us because they operated only on older Mac OSs. We looked into Wii and Wiimote emulators as well, but similar to the programs they were also broken due to not being updated in years.

When connecting to a computer, we realized there was another issue. When trying to connect, a PIN would be requested for us to enter. Whatever we entered didn't seem to matter, it would never authenticate. We tried all the different variations of what the PIN could be, using what we learned in our research about the Wiimote's Bluetooth, but nothing worked. We were confused at first, because when looking into how to connect a Wiimote with a computer a PIN password was never mentioned. We looked into this, but what we found seemed to imply that there was an OS update that led to this issue, which makes sense and connects back to the same reason why the programs that didn't work for us: technology has changed.

We were able to observe the traffic between the Wii and Wiimote, but we unfortunately were not able to gather much information. The messages were encrypted, which makes sense given that they were using Bluetooth. However, it posed a challenge that we were unable to solve for our deliverable.

In summary, many of our challenges can be tied back to one key limitation: time. A lot of time has passed since the Wii and Wiimote were commonly used. Technology has evolved and adapted, but they are remnants of a time when technology was just reaching its stride. The Wii and Wiimote are no longer made and aren't as popular as a result of newer, more advanced technology. This limited our options in addressing our challenges, and we recognize it limits the usefulness of future work.

4.2 Future Work

As discussed in section 3, we were able to passively and actively eavesdrop on the two communicating devices: the Wii Remote and the Wii Console. The next steps for performing the attack were to spoof the Wii Remote and relay the connection to the Wii Console. We provided a theoretical outline in that section, but here we will include some details about challenges we would expect to face in the implementation.

We mention that the Bluetooth addresses of the Wii Remote and Wii Console must be spoofed. This can be done easily with a Kali Linux tool called Spooftooph [8]. A second problem that might arise from that is deriving the link key, however from our research, it appears that the link key is derived from the Bluetooth PIN and can be reused throughout communication [6] [10]. To perform a successful man in the middle attack, we must be able to understand the packets between the Wii Remote and the Wii Console. The packets are encrypted with the link key as mentioned before. If we can derive the key, we can supply it in Wireshark and read the data. Finally, we run into the issue that the Wii Remote can only authenticate with one device at a time. To overcome this, we would have to wait for the Wii Remote to reauthenticate and enter discoverable mode and beat the legitimate Wii Console to responding to the Require Authentication message. This could be challenging if the Wii Remote and Wii Console are very close in proximity because the attacker would have to be at least as close as the Wii Console, theoretically. We have not devised a plan to overcome this except to assume the Wii Remote might enter discoverable mode at any time, and we could constantly send the Authentication Accepted messages to it, hoping they are received before the Wii Console's message.

4.3 Conclusion

Our main objective for this project was to find out if it's possible to "hack" a Wii Remote. When we decided on the topic of the project, we did not define the term "hack" because we did not know anything about the communication protocols involved. After sufficient research and gaining a deeper knowledge of Bluetooth protocols, we devised a plan to eavesdrop on the Wii Remote to Wii Console traffic, spoof a legitimate device, and then perform a man in the middle attack between the two devices. Although we could not fully implement all of these steps, we

learned that hacking a Wii Remote is possible. We believe with more time we could have been successful in these efforts.

The computer science track at Northeastern offers a deep foundation of software architecture and development, but the same cannot be said for hardware. During this project, we expanded our knowledge of how wireless hardware works and interacts with software. To be specific, we saw how short command bytes can be sent via Bluetooth to the Wii Remote and cause physical changes like the rumble feature or produce sound. Additionally we learned about a new wireless development system which was the Ubertooth One. It was useful for this project to sniff the Bluetooth signals and now we know a lot about its capabilities and what it could be used for in future projects.

Acknowledgements

Overall, we were saddened by the recent events which limited and caused shifts in our project. That being said, we believe the research we completed leading up to that point provided us with the foundation to outline a detailed theoretical solution that achieves our original deliverable. We are grateful to the professor and the teaching assistants for helping us choose our project and allowing us to redefine our goals when the pandemic interfered. We also thank them for helping us to procure hardware necessary for our project.

References

[1] “Bluetooth Sniffing with Ubertooth: A Step-by-step guide,” Bluetooth Sniffing with Ubertooth: A Step-by-step guide - Embedded Lab Vienna for IoT & Security, 05-Mar-2020. [Online]. Available:https://wiki.elvis.science/index.php?title=Bluetooth_Sniffing_with_Ubertooth:_A_Step-by-step_guide. [Accessed: 17-Apr-2020].

[2] D. Filizzola, S. Fraser, and N. Samsonau, “Security Analysis of Bluetooth Technology.” [Online]. Available:https://courses.csail.mit.edu/6.857/2018/project/Filizzola-Fraser-Samsonau-Bluetooth.pdf. [Accessed: 17-Apr-2020].

[3] K. Jang, J. Park, T.-J. Lee, and Y. Kim, “Reliable delivery of broadcast packet in Bluetooth,” IEEE VTS 53rd Vehicular Technology Conference, Spring 2001. Proceedings (Cat. No.01CH37202), May 2001.

[4] M. Ossmann, D. Spill, and M. Ryan, etc. “greatscottgadgets/ubertooth,” GitHub, 01-Dec-2019. [Online]. Available: https://github.com/greatscottgadgets/ubertooth. [Accessed: 17-Apr-2020].

[5] M. Ryan, “UBERTOOTH-RX,” ubertooth-rx(1) - ubertooth - Debian experimental - Debian Manpages, 01-Apr-2017. [Online]. Available: https://manpages.debian.org/experimental/ubertooth/ubertooth-rx.1.en.html. [Accessed: 17-Apr-2020].

[6] K. A. Scarfone and J. Padgette, “Guide to bluetooth security,” NIST Special Publication 800-121, 2008.

[7] D. Spill and A. Bittau, “BlueSniff: Eve meets Alice and Bluetooth.” Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.565.6674&rep=rep1&type=pdf

[8] “Spooftooph,” Penetration Testing Tools. [Online]. Available: https://tools.kali.org/wireless-attacks/spooftooph. [Accessed: 17-Apr-2020].

[9] Thierryzoller, “BTcrack-Bluetooth-PIN-Cracker-,” GitHub, 24-May-2019. [Online]. Available: https://github.com/thierryzoller/BTcrack-Bluetooth-PIN-Cracker-. [Accessed: 17-Apr-2020].

[10] “Wiimote,” WiiBrew, 15-Aug-2019. [Online]. Available: http://wiibrew.org/wiki/Wiimote. [Accessed: 17-Apr-2020].

[11] “Wiimote Project,” WiiMote Project, 16-Mar-2008. [Online]. Available: https://www.wiimoteproject.com/applications/getting-started/. [Accessed: 17-Apr-2020].