A Proof of Concept Denial of Service Attack Against Bluetooth IoT Devices

Shane Ditton*, Ali Tekeoglu†, Korkut Bekiroglu‡, Seshadhri Srinivasan§
*Network Computer Security Department, SUNY Polytechnic Institute, dittons@sunypoly.edu
†University of New Brunswick, Canadian Institute for Cybersecurity, ali.tekeoglu@unb.ca
‡Electrical Engineering Department, SUNY Polytechnic Institute, korkut.bekiroglu@sunypoly.edu
§Berkeley Education Alliance for Research in Singapore, seshadhri.srinivasan@bears-berkeley.sg

***Abstract*—Bluetooth technologies have widespread applications in personal area networks, device-to-device communications and forming ad hoc networks. Studying Bluetooth devices security is a challenging task as they lack support for *monitor* mode available with other wireless networks (e.g. 802.11 WiFi). In addition, the frequency-hoping spread spectrum technique used in its operation necessitates special hardware and software to study its operation. This investigation examines methods for analyzing Bluetooth devices' security and presents a proof-of-concept DoS attack on the Link Manager Protocol (LMP) layer using the InternalBlue framework. Through this study, we demonstrate a method to study Bluetooth device security using existing tools without requiring specialized hardware. Consequently, the methods proposed in the paper can be used to study Bluetooth security in many applications.**

***Index Terms*—Bluetooth, Internet of Things, Security, Denial of Service.**

I. Introduction

Bluetooth technology is one of the most prominent wireless connectivity options for the resource constraint Internet of Things (IoT) devices. Reasons for the popularity of Bluetooth among IoT include; low power consumption, lightweight networking, and ubiquity of the technology for interconnecting to other IoT devices. The most recent specification of Bluetooth is v5.1[1].As mentioned in the Bluetooth market update for 2019, ABI Research lists the primary markets for Bluetooth as mobile phones, tablets & PCs, audio & entertainment devices, automotive industry, smart buildings, smart industrial infrastructure, smart homes, and smart cities. They are expecting 48 billion Internet-enabled devices to be installed by 2021. Out of these IoT devices, about one-third will include Bluetooth technology for wireless communication.

With the ubiquitous deployment of Bluetooth technology embedded within billions of IoT devices, comes the security risks associated with it [1]. In particular, it is being deployed in a number of safety-critical products, including medical devices and connected automobiles. Recently, a brain-machine interface (BMI) start-up company, Neuralink, has announced that Bluetooth technology will be utilized to connect a specialized chip implanted into the human brain to connect it to a controller device, most likely a mobile application, which in turn connects to the Internet [2]. Bluetooth enabled insulin pumps, blood glucose monitoring systems, have been in the market for a while now. Around 2011, they have been shown to be vulnerable for multiple security issues in Bluetooth stack, which could let an attacker inject fatal doses remotely through the exploited insulin pump. Security of these Bluetooth enabled medical IoT devices is of utmost importance due to the potentially fatal consequences after the exploitation of vulnerabilities in the Bluetooth stack. Moreover, Bluetooth devices carry significant personal information which is critical in many ways.

Recently, with a constant, on-board Internet connection, cars became massive mobile IoT devices on wheels. The automobile industry has been embedding Bluetooth stack into vehicles for about a decade now. The attack surface on automobiles has been exponentially increasing with the surge in the integration of computer sub-systems. These sub-systems form controller area networks (CAN), and an exploit in one sub-system, could spread to the other which would eventually lead to complete take-over of control of a vehicle. Bluetooth security becomes significantly important in this case as well to prevent fatal malicious attacks.

In a previous work [3], we have analyzed the security of Bluetooth Low Energy (BTLE) and detailed an approach to test for security issues of BTLE enabled devices with low-cost off-the-shelf devices and open-source software, such as Ubertooth and Bluefruit. In this paper, we present details of the design and implementation of a proof-of-concept denial of service attack against Bluetooth enabled devices, utilizing Bluetooth's Link Manager Protocol (LMP). First, we give a brief overview of the recent literature on Bluetooth security research in Section II. In Section III, an overview of Bluetooth architecture followed by different approaches to capturing Bluetooth network traffic for security research purposes are presented. Then, in Section IV we are going to explain the design and tools we used for proof-concept Denial of Service attack. In Section V, we will present the implementation and experiment details. And finally, the conclusion and future work will be presented VI.

II. Related Work

In this section, recent research from the literature on the security and privacy of Bluetooth connected IoT devices,

[†]This work is supported by State University of New York Polytechnic Institute Undergraduate Research Program (SURP) during summer 2019.

[1]https://www.bluetooth.com/specifications/bluetooth-core-specification/

is presented to the best of our knowledge. There has been significant attention on Bluetooth technology with the proliferation of IoT devices, which stimulated the interest of security researchers from academia as well as industry. Researchers have been reporting a variety of Bluetooth related potential attacks and vulnerabilities since the popularity of Bluetooth picked up.

The most recent version of Bluetooth technology standard v5.0 is investigated by security researchers in [4]. They experimented with Secure Simple Pairing (SSP) protocol which is responsible for the pairing of devices. SSP contains a security mechanism called the passkey entry association model. The authors analyzed this model and found that the passkey entry association model is vulnerable to Man-in-the-Middle attacks, as soon as the host re-uses the passkey. As a solution, they proposed a new protocol to fix this vulnerability in the passkey entry model of SSP in Bluetooth v5.0. Authors emphasized the importance of security in Bluetooth v5.0, due to its commonplace use in home networking systems. They underline that people use home IoT devices that are utilizing Bluetooth, usually transfer personal, sensitive information.

Researchers find a potential weakness in the initial phase of pairing which would lead to impersonation of a device [5]. This vulnerability requires a sniffer to capture the Input-Output capability and public keys during the initial phase of pairing. Potential problem exists even with Bluetooth version 5.0. To counter the potential problem, researchers developed an enhancement to SSP by embedding two new security levels. They attempted to reduce the potential for man-in-the-middle attacks during the pairing phase.

In recent years, System-on-Chip (SoC) designs became popular. An SoC system embeds multiple components of computer electronics in a single board/chip. For instance, Bluetooth and WiFi electronics could be integrated into a single chip. Security researchers discovered two chip-level vulnerabilities, nicknamed BleedingBit [6], in SoC chips manufactured by Texas Instruments that have integrated BLE and WiFi components. With these vulnerabilities, an attacker would gain unauthenticated access to remote-code-execution on the chip. What makes it even more severe is this chip very commonly used in Cisco, Meraki and Aruba enterprise level access points. That means an exploit in the BLE component of the chip would let the attacker compromise the enterprise WiFi network as well.

A group of serious security issues was discovered in implementations of the Bluetooth stack [7], which was named "BlueBorne". These vulnerabilities affected a variety of operating systems, including Android, iOS, Windows and Linux [8] as well as a desktop and IoT based devices. Bluetooth devices that are not in discoverable mode or devices that are not paired with the attacker are still vulnerable for "BlueBorne" attack unless they are patched. In [9], security researchers investigated Bluetooth protocol's implementation in the Android system and uncovered design flaws which could lead to serious security issues. One of the findings was in authentication, for instance, if a paired device has a change in its profile, other ends of the pairing is not notified, and trust is still maintained. Bluetooth's omnipresence in home automation and entertainment systems urged researchers to investigate its security model in more detail [10]. They looked into v5.0, the latest standard and proposed a formal security model to evaluate secure simple pairing (SSP) protocol.

A survey of recent Bluetooth exploits and the recommended mitigation techniques are presented in [11]. Authors mention the recent popularity of Bluetooth, low-cost, low-power, short-range wireless technology, especially amongst Internet-of-Things (IoT) devices. Real-life examples of successful exploits are discussed and the importance of understanding attack risks faced with using Bluetooth technology within our daily used devices is presented in this study.

Furthermore, the authors in [12] investigate major security issues in Bluetooth protocols with the help of a Man-in-the-Middle (MitM) attack setup. The investigation tests the proposed attacks with necessary equipment after introducing some of the well-known tools for Bluetooth hacking. The work proved that how easy to interfere with the Bluetooth device to control the data and even the mobile devices in some cases.

Apple products are no exception for Bluetooth popularity, on the contrary, almost all Apple devices including; Apple Watch, iPods, Apple-TV, iPhones have Bluetooth capability. Researchers [13] examined communication between iOS and macOS devices. One protocol for keeping connections alive under Bluetooth stack is Bluetooth Low Energy Continuity protocol, which is designed for inter-operability and seamless experience for users. However researchers, in their experiments, found that some fields in the Continuity protocol network packets are transmitted in clear text, which leaks information about users that would enable malicious entities to track their devices. One such information is the MAC address of the device, which is designed to be changing randomly for increased privacy. However, due to discovered flow, attackers can figure out the MAC address with the help of the Continuity protocol's clear text data-fields and profile and track Apple users.

III. Background

Before going into the details of the proposed proof-of-concept Denial of Service attack, in this section, we provide some background information on Bluetooth architecture and different approaches to investigate Bluetooth packets for security research.

A. Overview of Bluetooth Architecture

Bluetooth devices operate at an RF-based 2.4 GHz frequency in the license-free, globally available ISM (Industrial, Scientific, and Medical) radio band. It was originally designed for short-range (i.e. up to 10 meters), connectivity technology & solution for portable, personal devices. The Bluetooth spec comprises an HW & SW protocol specification usage case scenario profiles and interoperability requirements as shown in Figure 1.

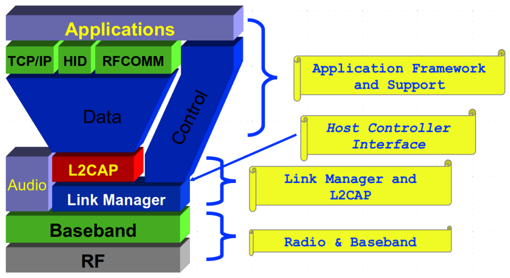


Fig. 1: Bluetooth architecture, and protocol layers

Worldwide availability and compatibility is the advantage of operating in this band. However, a potential disadvantage is that Bluetooth devices must share this band with many other RF emitters such as automobile security systems, other wireless communications standards (such as 802.11), and ordinary noise sources (such as microwave ovens). To overcome this challenge, Bluetooth employs a fast frequency-hopping scheme and uses shorter packets than other standards in the ISM band. This scheme makes Bluetooth communication more robust and more secure.

B. *Capturing HCI Messages*

The Host Controller Interface (HCI) messages provide a simple way to analyze Bluetooth devices. The communication between the host operating system and the physical Bluetooth device is managed by the HCI subsystem within the Bluetooth stack. Since these messages are visible to the operating system, they can be captured through the use of software on the host.

For instance, in the Android mobile operating system, this can be accomplished by enabling ”the Bluetooth HCI snoop log” in Android Developer Options. Then, Android would write all incoming-outgoing Bluetooth packages on the mobile device into a *.pcap* formatted file, which could be extracted from the phone (i.e. using ADB tool) and examined offline on a computer with a Wireshark network packet analyzer. On Linux systems using the BlueZ Bluetooth stack, `hcidump`, `tcpdump`, or `Wireshark` can be used to capture HCI messages for further analysis. This allows a researcher to see everything happening above the HCI layer. This may be sufficient for some types of research. However, if one wants to see into the lower layers of the Bluetooth stack, this approach with HCI snooping will not suffice.

C. *Hop Sequence Following Sniffers*

One option for viewing the lower layers of the Bluetooth protocol is to sniff the packets out of the air. The main obstacle to this strategy is the FHSS technique that Bluetooth uses to avoid interference. Bluetooth Classic operates over 80 channels, with each channel being 1 MHz wide, while Bluetooth Low Energy operates over 40 channels, with each channel being 2 MHz wide. There are two main approaches to solving this problem. The first is to record all data on all channels. This is sometimes referred to as ”wideband” capture, Wideband capture can be achieved using Software Defined Radios (SDR). Several commercial protocol analyzers also use this technique to capture Bluetooth traffic. The SDR approach can be implemented with open-source software such as GNU-Radio. One drawback for SDR is the amount of data produced by sniffing the whole Bluetooth spectrum, and the amount of CPU power required for digital signal processing on this raw data [14].

The second approach is to try to ”follow” a connection by hopping to the correct channel at the correct time. This is the approach used by many low-cost USB connected Bluetooth sniffers, such as the Adafruit’s Bluefruit and the Ubertooth, as shown in Figure 2. Connection following avoids the need to use an expensive radio receiver with 80 MHz of instantaneous bandwidth, but it introduces several challenges and limitations. First, these sniffers can only listen to one channel at a time. Therefore, only one Bluetooth network may be monitored at a time. Second, the sniffer must hop very precisely to avoid dropping Bluetooth packets. In practice, it can be challenging to get a high-quality packet capture with these devices.

D. *Firmware Reverse Engineering*

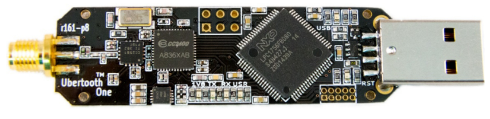
Another technique for obtaining low-level details about the operation of a Bluetooth chip is to reverse engineer its firmware. Dennis Mantz, a researcher from TU Darmstadt, achieved this with the BCM4339 in his thesis titled ”InternalBlue - A Bluetooth Experimentation Framework based on Mobile Device Reverse Engineering” [15]. The framework that was developed is compatible with all Broadcom chips, with varying levels of support [16]. The InternalBlue framework uses vendor-specific HCI commands to gain access to the RAM and ROM of the Broadcom Bluetooth controller, which allows the framework to control the execution of the chip and to retrieve interesting information [17]. In particular, InternalBlue supports capturing Link Manager Protocol (LMP) packets. LMP can be ingested and parsed by Wireshark through the use of a modified third-party Wireshark dissector. LMP packets can also be injected into active connections using the InternalBlue framework. This process can be used for fuzzing at the LMP layer, or to implement various attacks on Bluetooth pairing.

IV. Tools and Methods

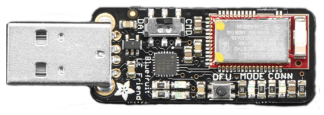
Firmware source code for Bluetooth chips is not shared publicly by chip manufacturers, even though the Bluetooth standard is openly available. For researchers interested in investigating the internal workings of Bluetooth firmware, there are multiple challenges. These challenges are as follows; (i) Low-level operations are handled by the Bluetooth controller, not by the host. (ii) The Host Controller Interface (HCI) limits access to the controller from the host. (iii) Sniffing Bluetooth RF signals can be difficult due to FHSS.

A. *Capturing Bluetooth at the RF Layer*

One approach is to follow a Bluetooth connection as it hops through the spectrum. Affordable sniffers (such as the Ubertooth and Adafruit Bluefruit) use this strategy.



(a) Ubertooth One



(b) Bluefruit Sniffer

Fig. 2: Ubertooth for open-source Bluetooth packet sniffing and Adafruit's BLE Sniffer Bluefruit

Another approach is to sniff the entire RF band used by Bluetooth. This can be done with a software-defined radio, provided that it has enough bandwidth. Commercial Bluetooth protocol analyzers that use this approach are also available. We ultimately opted to pursue an alternative approach.

B. InternalBlue Framework

In 2018, development began on a framework called ”InternalBlue”, as part of a master's thesis at TU Darmstadt. Development has continued at TU Darmstadt's Secure Mobile Networking Lab (SEEMOO). InternalBlue uses Broadcom-specific HCI commands to gain access to the firmware of Broadcom Bluetooth controllers. Using this framework, the firmware can be examined, controlled, and modified while the chip is running. InternalBlue can also be used for live/dynamic analysis through the use of a ”breakpoint” feature on supported chips. We used the framework on a Nexus 5 and on the Raspberry Pi 3.

C. Ghidra: Reversing Bluetooth Firmware for Patching

In conjunction with InternalBlue, we also used Ghidra, a software reverse engineering suite developed by the US National Security Agency. We used Ghidra to perform static analysis on firmware images that we obtained with InternalBlue. This was used to develop patches for the firmware.

V. Experiments: Attacks and Mitigations

It has been shown previously that one can start a DDoS attack against a Bluetooth device in proximity with simple Linux command-line tools. For instance, an attacker in proximity could start reconnaissance with `hcitool` and can scan the environment for Bluetooth devices (i.e: $ hcitool scan). If there are any, the MAC address would be found out. With the knowledge of the target device's MAC, the next step would be to start ping with large sized request packets in flood mode. `l2ping` tool could be used for this step (i.e: $ l2ping -i hci0 -s 600 -f <Bluetooth MAC>). If the attacker has multiple Bluetooth interfaces in his attacking machine, with automated scripts, he can start a devastating DDoS attack on any target in proximity. However, this approach is brute-force and it would generate an immense amount of packets potentially cause congestion in the 2.4GHz frequency domain. Instead of brute force attack, in this research, we used a different approach that would not generate a flood of packets directed towards the victim. We explain the concept of filling the available connection slots in the LMP layer on the target device below. Our test-bed for the proof-of-concept attack is depicted in Figure 3.

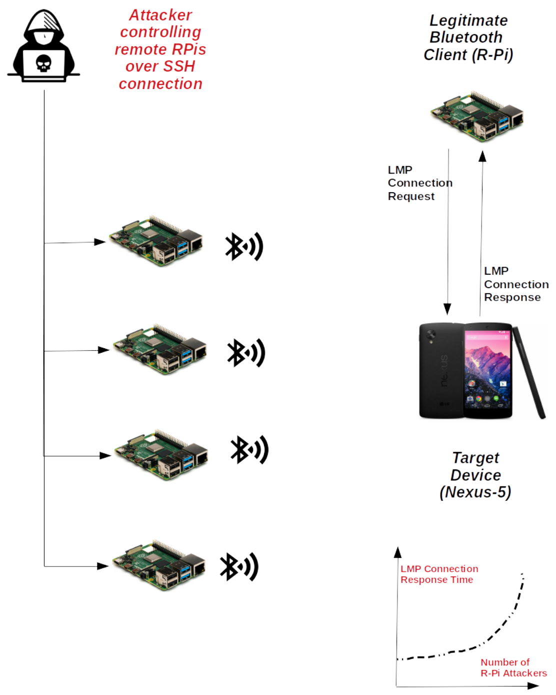

Attacker controlling remote RPis over SSH connection
Legitimate Bluetooth Client (R-Pi)
LMP Connection Request
LMP Connection Response
Target Device (Nexus-5)
LMP Connection Response Time
Number of R-Pi Attackers

Fig. 3: Test-bed for PoC LMP denial of service attack

A. Proof of Concept LMP DoS Attack

In the thesis [15], the author identified a connection tracking data structure in the firmware of the BCM4339 (used on the Nexus 5). This data structure keeps track of each of the active LMP connections and has a maximum capacity of 12 connections. Our concept was to fill this table with spurious LMP connections from one or more devices, preventing the Nexus from connecting to any other devices.

1) Keeping an LMP session open: In our testing, different devices exhibited different behaviors when it came to how they handled LMP connections. The Nexus 5, when initiating connections, tended to maintain connections until it received a ”detach” packet from the other device.The Raspberry Pi 3 would almost immediately send a detach packet and disconnect. This made the DoS attack difficult to execute using the Raspberry Pi 3.

In order to resolve this, we developed a firmware patch for the Raspberry Pi 3 which prevented detach packets from being sent (see Algorithm 1). We did this by referencing [15] the author's work on reversing a function he calls “send_lmp_packet”. This function takes a “connection handle” (or connection number) and a pointer to packet data as arguments.

Our patch hooks the send_lmp_packet function, branching into our code whenever an LMP packet arrives to be checked and sent. We then inspect an opcode byte within the packet buffer (referenced by the pointer argument), looking for a detach packet. If a detach packet is found, we branch into a handler for invalid LMP packets. Otherwise, we branch back into the main send_lmp_packet function and continue as normal.

Algorithm 1 Setting up connection to HCI interface

```
    Libraries:
    from pwn import *
    from internalblue.adbcore import ADBCore

 1: internalblue = hcicore()
 2: internalblue.interface=internalblue.device_list()[0][1]
 3: if not internalblue.connect():
        log.critical("No connection to target device.")
        exit(-1)
 4: HOOK_ADDRESS = 0x170ba
 5: ASM_ADDRESS = 0x207eee
 6: ASM_SNIPPET = """
         b prevent_detach
         prevent_detach:
         mov r4, r1
         ldr r0, [r1, #12]
         and r0, 0xff
         cmp r0, 0x0e
         beq 0x1712a
         b 0x107be
        """
 7: progress_log = log.info("Writing prevent_detach to RAM...")
 8: code = asm(ASM_SNIPPET, vma=ASM_ADDRESS)
 9: if not internalblue.writeMem(address=ASM_ADDRESS,
        data=code, progress_log=progress_log):
        log.critical("Error writing to RAM.")
        exit(-1)
10: log.info("Writing send_lmp_packet hook to ROM...")
11: patch = asm("b 0x%x" % ASM_ADDRESS,
    vma=HOOK_ADDRESS)
12: if not internalblue.patchRom(HOOK_ADDRESS, patch):
        log.critical("Error patching ROM.")
        exit(-1)
13: internalblue.shutdown()
14: log.info("Goodbye!")
```

We initially developed our patch for the Nexus 5 to take advantage of InternalBlue features such as tracepoints. We then later ported them to the Raspberry Pi 3. The patches were loaded onto the devices using the InternalBlue framework.

2) *Measuring Victim's System Vitals under DoS attack:* Our experiments included a rooted Nexus 5 phone as the victim device. The victim could be any Bluetooth enabled device, however in order to monitor the effects of our DoS attack better, we opted to use an Android device. In Android OS, texttt"dumpsys"[2] is a command-line tool that could be executed through Android Debugging Bridge to measure system information. This tool is designed to monitor many different system vitals, from CPU to battery usage. We used `"ADB shell dumpsys cpuinfo"` to measure the system's CPU statistics, before, during and after the proposed Bluetooth DoS attack. Figure 4 plots the CPU information from the Nexus 5.

[2]https://developer.android.com/studio/command-line/dumpsys

The dotted red line on the top shows the total CPU usage, which shows that our DoS attack does not affect the CPU usage on the victim. This proves that the proposed method is not a brute force attack, on contrary it is lightweight on the victim, which makes it much harder to detect the anomaly on the victim's side.

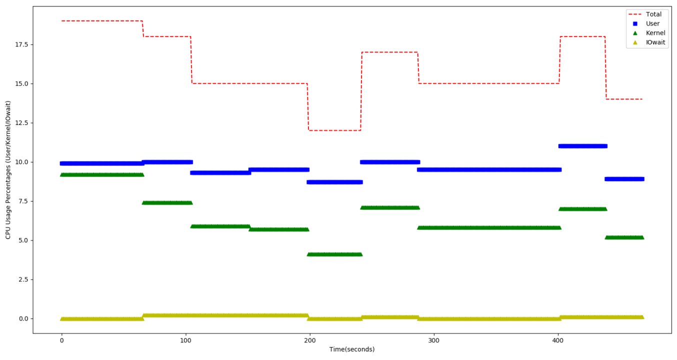


Fig. 4: CPU usage on the victim (Nexus 5) is not affected during the LMP denial of service attack

During the experiment in test-bed (depicted in Figure 3), we captured Bluetooth traffic with Wireshark with the help of InternalBlue framework, on the victim device. First, a legitimate user (R-Pi on the top of Figure 3) is connected to Nexus 5. Since both devices are not peripherals, after exchanging profile information, they decide to disconnect gracefully. We did this legitimate connect-disconnect process before the attack, with one attacker, with 2 attackers, with 3-attackers, and with 4-attackers. When the number of attackers reached 4, a connection request from legitimate R-Pi to victim Nexus 5 timed-out, and failed. Which proved that even though the number of LMP slots in Bluetooth stack is 12, in practice Nexus 5 was only able to handle 3 simultaneous LMP connections. Figure 5 shows the delta time between the first connection request packet and the final connection teardown request packet, without and with the DoS attack.

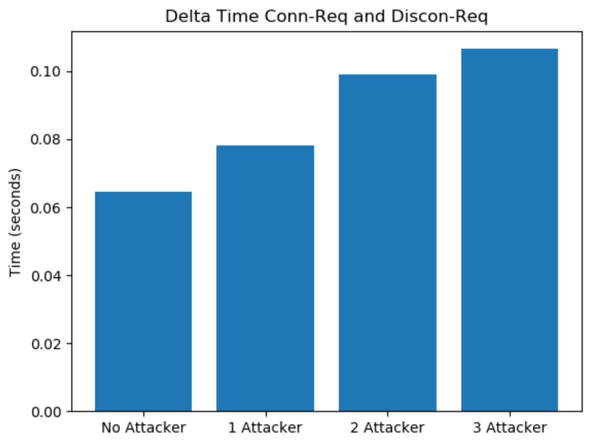


Fig. 5: The amount of time it takes from the initial connection request packet to the final disconnection request packet between Nexus 5 and legitimate R-Pi clients. Measurements are taken before and during the LMP denial of service attack. When the number of attackers reached 4, a connection request from legitimate devices times out.

One possible strategy to mitigate this attack would be to add some type of exponential back-off timer to prevent devices from repeatedly establishing LMP connections which do not result in the pairing. Another strategy would be to give the user more control over when and how LMP connections are established. Currently, no user-perceptible feedback is provided, and the conditions under which LMP connections are established and maintained varies by device.

3) Single Device DoS attack: We also investigated whether or not it was possible to conduct a denial of service using one device. This would make the attack more practical. Using [15] again as a guide, we examined the connection data structure. We then used Ghidra's cross-reference feature to find functions which reference this piece of memory. Through a thorough review using breakpoints and manual analysis, we found a function that checks if a connection already exists before establishing it. We modified this function to skip this check.

After being modified, the link manager attempted to create another connection, but this caused an error. We surmise that the baseband controller, which is implemented as a separate chip on this Bluetooth controller, is also checking for an existing connection. Since this chip is not accessible via InternalBlue, we could not establish additional connections.

4) LMP Fingerprinting & Mitigations to Improve Privacy: One of the major findings in SEEMOO's InternalBlue research is that devices provide identifying information through LMP. This includes (but is not limited to): (i) Device name, (ii) LMP version and subversion, (iii) Manufacturer ID. This information can be used to profile a target for vulnerabilities, or for device tracking purposes. This is a potential privacy and security concern.

A potential mitigation against LMP layer fingerprinting, and privacy protection could be achieved as follows. By hooking the "send_lmp_packet" function, we are able to modify LMP packets in memory as they're dispatched by the Bluetooth controller. This allows us to modify or conceal identifying information.

VI. Conclusion and Future Work

This paper provided an approach to study Bluetooth security using the Link Manager Protocol (LMP) layer without using specialized hardware. We demonstrated the approach through a Denial of Service (DoS) attack using the InternalBlue framework. The proposed method had three operations to study the security, they are: (i) Monitor LMP traffic (ii) Inject arbitrary LMP packets (iii) Modify or reject LMP packets on the fly. Our study shows that the proposed approach can be used to perform a hard to detect Denial of Service attack.

Future work may focus on finding a way to conduct the LMP denial of service using only one attacker device (i.e. one Raspberry-Pi armed with multiple usb Bluetooth dongles), and on identifying the cause of the LMP connection timeouts that we observed. Additional exploration of the Bluetooth controller's firmware may also yield new security findings.

References

[1] K. Haataja, K. Hyppnen, S. Pasanen, and P. Toivanen, *Bluetooth Security Attacks: Comparative Analysis, Attacks, and Countermeasures*. Springer Publishing Company, Incorporated, 2013.

[2] E. Musk, "An Integrated Brain-Machine Interface Platform with Thousands of Channels," *bioRxiv*, 2019. [Online]. Available: https://www.biorxiv.org/content/early/2019/07/18/703801

[3] S. Sevier and A. Tekeoglu, "Analyzing the Security of Bluetooth Low Energy," in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, Jan 2019, pp. 1–5.

[4] D.-Z. Sun, Y. Mu, and W. Susilo, "Man-in-the-middle attacks on secure simple pairing in bluetooth standard v5.0 and its countermeasure," *Personal Ubiquitous Comput.*, vol. 22, no. 1, pp. 55–67, Feb. 2018. [Online]. Available: https://doi.org/10.1007/s00779-017-1081-6

[5] S. Gajbhiye, S. Karmakar, M. Sharma, and S. Sharma, "Bluetooth secure simple pairing with enhanced security level," *Journal of Information Security and Applications*, vol. 44, pp. 170 – 183, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214212618301728

[6] B. Seri, G. Vishnepolsky, and D. Zusman, "Bleeding Bit: The Hidden Attack Surface within BLE Chips," Armis, Tech. Rep., 2018. [Online]. Available: https://go.armis.com/hubfs/BLEEDINGBIT-TechnicalWhitePaper.pdf

[7] B. Seri and G. Vishnepolsky, "BlueBorne: The dangers of Bluetooth implementations: Unveiling zero day vulnerabilities and security flaws in modern Bluetooth stacks," Armis, Tech. Rep., 2017. [Online]. Available: https://go.armis.com/hubfs/BlueBorneTechnicalWhitePaper-1.pdf

[8] B. Seri and A. Livne, "Exploiting BlueBorne in Linux-based IoT devices," Armis, Tech. Rep., 2019. [Online]. Available: https://go.armis.com/hubfs/ExploitingBlueBorneLinuxBasedIoTDevices.pdf

[9] F. Xu, W. Diao, Z. Li, J. Chen, and K. Zhang, "BadBluetooth: Breaking Android Security Mechanisms via Malicious Bluetooth Peripherals," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/badbluetooth-breaking-android-security-mechanisms-via-malicious-bluetooth-peripherals/

[10] D.-Z. Sun and L. Sun, "On Secure Simple Pairing in Bluetooth Standard v5.0-Part I: Authenticated Link Key Security and Its Home Automation and Entertainment Applications," *Sensors*, vol. 19, no. 5, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/5/1158

[11] A. M. Lonzetta, P. Cope, J. Campbell, B. J. Mohd, and T. Hayajneh, "Security Vulnerabilities in Bluetooth Technology as Used in IoT," *Journal of Sensor and Actuator Networks*, vol. 7, no. 3, 2018. [Online]. Available: https://www.mdpi.com/2224-2708/7/3/28

[12] T. Melamed, "An Active Man-in-the-Middle Attack on Bluetooth Smart Devices," *International Journal of Safety and Security Engineering*, vol. 8, no. 2, 2018. [Online]. Available: https://www.witpress.com/Secure/ejournals/papers/SSE080202f.pdf

[13] J. Martin, D. Alpuche, K. Bodeman, L. Brown, E. Fenske, L. Foppe, T. Mayberry, E. Rye, B. Sipes, and S. Teplov, "Handoff All Your Privacy - A Review of Apple's Bluetooth Low Energy Continuity Protocol," *Proceedings on Privacy Enhancing Technologies*, vol. 1, no. 4, pp. 34–53, 2019. [Online]. Available: https://content.sciendo.com/view/journals/popets/2019/4/article-p34.xml

[14] W. Liu, D. Pareit, E. De Poorter, and I. Moerman, "Advanced spectrum sensing with parallel processing based on software-defined radio," *EURASIP JOURNAL ON WIRELESS COMMUNICATIONS AND NETWORKING*, p. 15, 2013. [Online]. Available: http://dx.doi.org/10.1186/1687-1499-2013-228

[15] D. Mantz, "InternalBLUE - A Bluetooth Experimentation Framework Based on Mobile Device Reverse Engineering," Master's thesis, Technische Universitat Darmstadt, Secure Mobile Networking Lab Department of Computer Science, July 2018.

[16] D. Mantz, J. Classen, M. Schulz, and M. Hollick, "InternalBlue - Bluetooth Binary Patching and Experimentation Framework," *CoRR*, vol. abs/1905.00631, 2019. [Online]. Available: http://arxiv.org/abs/1905.00631

[17] J. Classen and M. Hollick, "Inside Job: Diagnosing Bluetooth Lower Layers Using Off-the-Shelf Devices," *CoRR*, vol. abs/1905.00634, 2019. [Online]. Available: http://arxiv.org/abs/1905.00634