Security Analysis of a Medical IoT Device: Data Leakage to an Eavesdropper

Stephanie L. Long
Air Force Institute of Technology
Stephanie.Long@afit.edu

Richard Dill
Air Force Institute of Technology
Richard.Dill@afit.edu

Barry E. Mullins
Air Force Institute of Technology
Barry.Mullins@afit.edu

Abstract

Embedded technology known as the Internet of Things (IoT) has been integrated into everyday life, from the home, to the farm, industry, enterprise, the battlefield, and even for medical devices. With the increased use of networked devices comes an increased attack surface for malicious actors to gather and inject data, putting the privacy of users at risk. This research considers the Masimo MightySat fingertip pulse oximeter and the companion Masimo Professional Health app from a security standpoint, analyzing the Bluetooth Low Energy (BLE) communication from the device to the application and the data leakage between the two. It is found that with some analysis of a personally owned Masimo MightySat Rx through the use of an Ubertooth BLE traffic sniffer, static analysis of the `HCI_snoop.log` *and application data, and dynamic analysis of the app, data could be reasonably captured for another MightySat and interpret it to learn user health data.*

1. Introduction

The Internet of Things (IoT) technology has greatly advanced in the past decade. Security Today reported 26.7 billion IoT devices in use, with an estimated 35 billion to be expected by 2021 and 75 billion by 2025 [1]. The amount of technological advancement and investment being put into these networked devices is shifting the way everyday life is conducted. While no official definition of the IoT exists, this paper adopts the following set forth by the Government Accountability Office, "the set of Internet-capable devices, such as wearable fitness devices and smartphones, that interact with the physical environment and typically contain elements for sensing, communicating, processing, and actuating" [2].

1.1. Background

The first conceptual usage of networked embedded technology was in 1982 when a group of computer science students connected a Coca-Cola machine to the network in order to monitor the supply of cold drinks [3]. From this idea, coined the term "Internet of Things" by Kevin Ashton during a presentation for Procter & Gamble [4]. Since then, IoT has risen in popularity in all facets of technology.

Industrial Control Systems (ICS) use IoT protocols to communicate in order to share system status updates and to be controlled remotely [5]. Residential homes and businesses have integrated wireless cameras to provide security, remote lighting controls to control light levels, and even sensors to monitor temperature and movement in an office to conserve energy [6]. The agricultural industry uses IoT devices to maintain soil pH levels for optimum crop growth and to monitor health and location of the animals [7]. Furthermore, hospitals and health care providers' IoT devices keep track of patient health data, and monitor the health of residential patients with portable IoT devices [8].

1.2. IoT Composition

Networked embedded devices differ from traditional computer systems in a few ways; due to their specialized functions and unique architectures, patient data and unauthorized access security is difficult [9]. IoT manufacturers employ custom operating systems and take novel approaches to developing IoT devices, often at the expense of security best practices. Many IoT devices are designed to be low-powered devices that conserve battery life, and due to their small nature, use low-speed CPUs and/or have limited memory. Due to these architectural constraints, the ability to use cryptographic algorithms is limited, which results in unencrypted data transmission [10].

IoT devices utilize a variety of communication protocols. Some of the most common include Wi-Fi, Bluetooth Classic, Bluetooth Low Energy (BLE),

URI: https://hdl.handle.net/10125/71448
978-0-9981331-4-0

Message Queuing Telemetry Transport (MQTT), and Zigbee [11].

1.3. Target Device & Protocol

This research looks at the Masimo MightySat Rx, serial number 1822689309 [12]. It is a fingertip pulse oximeter that records the oxygen saturation, pulse rate, perfusion index, and pleth variability index of the wearer. The device uses BLE as the communication protocol. In order to understand the security implications of BLE, a brief explanation of how the protocol works is required.

Bluetooth Low Energy is a lightweight, low-power protocol that is used to transmit small amounts of data from a peripheral device, such as the Masimo MightySat Rx, to a central device, such as a smartphone [13]. The BLE protocol stack is unique (Figure 1), and the important components to understand in this research are the Generic Access Profile (GATT) and the Attribute Protocol (ATT). GATT is the general specification used for sending and receiving attributes. The ATT then exists below the GATT and Generic Attribute Profile on the protocol stack. Each attribute consists of a unique 128-bit string ID known as a Universally Unique Identifier (UUID). ATT is responsible for transporting services and characteristics. A characteristic is a single value and contains a descriptor to describe that value. A service is a collection of characteristics, such as a heart rate monitor which would include characteristics such as the heart rate measurement [14].

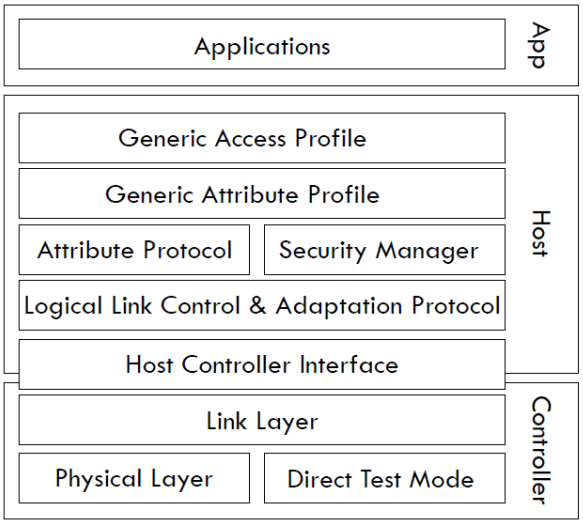

Figure 1. The BLE Protocol Stack [14]

1.4. BLE Attack Surface

BLE is susceptible to several different types of attacks, but the most pervasive are passive eavesdropping and a man-in-the-middle (MITM). Passive eavesdropping can be accomplished by a third-party entity monitoring the traffic transmissions between devices. If this information is sent in clear text, credentials and command text can be easily gleaned. Even if the data is encrypted, a clever attacker can use clues present in the packets to crack the key, dependent on the cryptographic scheme used. A MITM attack occurs when passive eavesdropping is taken a step further and the attacker intercepts the data between the devices and acts as a proxy, making himself a middle-man and posing as the peripheral device to the central and vice versa. The attacker can modify the traffic and inject malicious code instead of the intended data from the peer device.

In the case of a device collecting medical data, information leakage violates the expectation of privacy. While it may not inherently seem dangerous to leak pulse rates or oxygen levels, this information may be used to diagnose an ongoing medical condition, and corrupt data could result in misdiagnosing a patient so they do not receive the proper treatment. OWASP lists insecure data transfer as one of the top ten vulnerabilities facing IoT devices [15]. Sniffing the data may alert the malicious actor to a medical issue that the user has that should be private, and may also allow the attacker the possibility to compromise the integrity of the data. This research seeks to determine the data leakage of the target device from the perspective of a security analyst.

1.5. Structure

The structure of this paper details related work in Section II, where it reviews research on the security of the BLE protocol and wearable IoT devices, as well as on dynamic analysis of applications. Section III generalizes the methodology employed in this research and the dependent tools. Section IV examines the results of this methodology applied to the Masimo MightySat Rx device and its corresponding manufacturer application. Sections V and VI then propose future work a researcher could do to further examine this device and other like devices, then concludes with key findings.

2. Related Work

2.1. BLE Security

Ryan's research on BLE was monumental in reverse engineering the relatively new protocol and evaluating its security [16]. He determined that the BLE Special Interest Group (SIG) chose to use a custom key exchange protocol instead of a well-known and well-trusted mechanism such as Elliptic Curve Diffie-Hellman (ECDH) key exchange. Using an Ubertooth [17], which ported the network traffic to Wireshark [18], Ryan captured data from several devices, though not specified. From these, he generalized the data and deduced that there were four unique values necessary to follow a connection; hop interval, hop increment, unique access address, and cyclic redundancy check (CRC) init, all sent in the protocol data unit (PDU) of a BLE Packet (Figure 2).

Table 1. BLE Packet Structure[16]

Preamble	Access Address	Protocol Data Unit (PDU)	CRC
2 Bytes	4 Bytes	2-257 Bytes	3 Bytes

The unique access address was observable in the packet data following the BLE standard access address, and the CRC init value was calculable from the CRC at the end of the packet. The hop interval and increment were both calculated based on observing the network traffic and the time between packets on a specific channel.

Table 2. Pairing Modes for BLE

Pairing Mode	TK Value
Just Works	0
6-Digit Pin	0-999,999
Out of Band	128 value exchanged out of band

With these values, Ryan determined that three keys were used to encrypt the conversation, the temporary key (TK), the short-term key (STK), and the long-term key (LTK). The TK is calculable based on knowledge of the pairing method possible values, shown in Table 1, for BLE by brute forcing the possibilities. Cracking the TK allowed the STK then LTK to be cracked and the communication to be decrypted.

Ryan also wrote firmware as a contribution from his research for the Ubertooth sniffer to automatically parse out the necessary data, calculate the needed values, and follow the connection of a specified MAC address. His findings are vital to further research on security for IoT devices using BLE, and to understanding the packet structure of data flowing on the network layer.

2.2. BLE Traffic Sniffing and Injection Attack

During his presentation at DEFCON 2016, Gutierrez outlined a process for hacking BLE using a sniffer to gather clear text data from a thermostat [19]. The thermostat, the HOBO MX1101, was the same model that the Salisbury Cathedral boasted of using to secure the environment for a historical artifact, the Magna Carta. He was able to find the password in plain text within the sniffed traffic, which was used to authenticate with the device and to control the temperature settings. Using this password and the packet structure for sending commands, Gutierrez was able to impersonate the app interacting with the device and even force a reset for the entire system. Furthermore, he was able to catch a firmware update for the device and determine the mechanism to administer an update. With this information, he remotely flashed new firmware to make the temperature gauge instead be a heart rate monitor. His research illustrates what effects data leakage allows a malicious actor to have on a device.

2.3. Wearable IoT Security

Research done in the following works both examine the security of wearable fitness and health devices. Haperin et al. analyzed an implantable cardioverter defibrillator (ICD) that communicated patient health data wirelessly [20]. The team determined that the medical information, such as the patient's name, date of birth and cardiac values, was sent unencrypted and could be intercepted by a malicious actor. Furthermore, the researchers were able to exploit a testing interface to replay communication to the ICD in order to induce a shock to the patient's heart. This attack could result in death to a victim of the man-in-the-middle attack, and the device had no security measures in place to stop such an attack. Rahman et al. analyzed the poor security design utilized in the Fitbit (version unspecified) that allowed an attacker to reverse engineer the protocol and inject false fitness data to the online tracker [21]. One of the security flaws the researchers discovered was that the network traffic showed user credentials were passed across the network in clear text upon authenticating with the software. Furthermore, the data for the log files were also sent unencrypted, giving access to the actual health data. The researchers then analyzed the possible attacks on the system and developed an attack framework against the device.

Wang et al. discovered wrist-wearable BLE IoT devices, such as fitness monitors, could be used to extrapolate Personal Identification Numbers (PIN) [22]. The researchers were able to determine the PINs based

on two methods: sniffing attacks and an internal attack by putting malware on the user's app device. Data collected from the accelerometer was directly mapped to the key pushed on an Automatic Teller Machine (ATM) machine or a door entry keypad. The data pulled from the devices and the development of a distance-estimation algorithm allowed the attackers to determine the user's PIN with an 80% accuracy upon a one-time key entry and 90% upon a three-time key entry.

2.4. Mobile Application Security

Chen et al. developed an automatic fuzzing framework to test memory corruption vulnerabilities in IoT devices [23]. Fuzzing is a security method that sends random data to a device in order to find edge cases that may crash the device or cause it to act in a way that the developers did not intend [24]. Because pulling firmware on IoT devices may not always be possible or the tools may not be available, the researchers chose fuzzing as an alternative to test the security of the IoT device. This still comes with challenges for access to the device and understanding the unique protocols that IoT devices use. The team determined that if they identified the functions in the companion application for the device, these functions were candidates for fuzzing. Using this knowledge, the researchers were able to use taint analysis, which identifies every source of user data and the various input combinations [25], to change the values and identify vulnerabilities that crash the IoT devices. This research demonstrates that the companion application for an IoT device can provide vital data to determining the security of the IoT device itself.

3. Methodology

The methodology used in this paper combines BLE protocol sniffing between the device and phone, analysis of the phone's Bluetooth log, and static and dynamic analysis of the mobile application. The methodology is broken into three main steps:

1. Network Traffic Collection
    - Use `lescan` [26] or LightBlue [27] to locate device of interest and the corresponding MAC address
    - Use Ubertooth [17] hardware with `ubertooth-btle` [17] command-line tool to follow connection and capture traffic
    - Analyze in Wireshark [18] and search for plain text data
2. Static Analysis
    - Use Mobile Security Framework (MobSF) [28] or `jadx` [29] to inspect Android Application Package (APK) of the application
    - Determine function(s) of interest
    - Collect `HCI_Snoop.log` using `adb` on Android phone & analyze in Wireshark
3. Dynamic Analysis
    - Use `Frida` [30] to hook into function(s) of interest
    - Pull pertinent values out of function(s) of interest and attempt to correlate Wireshark capture data

Table 2 includes the tools required in the methodology and a brief description.

Table 3. Toolset Used in Methodology

Tool	Description
Ubuntu v18.04	Linux operating system
`lescan` [26]	Part of BlueZ protocol stack to scan for BLE devices
Ubertooth [17]	Wireless development platform for BLE signal collection
`ubertooth-btle` [17]	Command-line tool to interface with Ubertooth device
Wireshark v2.6.10 [18]	Open source network protocol analyzer
LightBlue v1.7.0 [27]	Mobile application for testing and simulation of BLE devices
`ADB` v1.0.39 [31]	Command-line tool for mobile debugging
`jadx` v11.0.7 [29]	Android dex and APK decompiler for Java source code
`Frida` v12.8.20 [30]	Dynamic code instrumentation toolkit for mobile apps
`MobSF` v3.0.9 [28]	Mobile application security assessment framework

This methodology is designed from the perspective of an attacker for what information might be gleaned without the victim's knowledge through data leakage between the device and the phone. A security analyst should determine this data leakage before an attacker has the opportunity to do so. Figure 3 provides the experimental configuration.

The static and dynamic sections assume that the phone must be rooted to allow the user full access to the phone and directories. This is required for dynamic analysis using `Frida` and to pull the relevant files in an efficient manner. The developer's options must also be configured to enable universal serial bus (USB) debugging and for the Bluetooth and BLE activity logs on the phone present in the host controller interface `HCI_snoop.log` file.

3.1. Network Traffic Collection as a Passive Listener

The research model employed here consists first of gathering data. The researcher must use the Ubertooth platform with the `ubertooth-btle` command line tool to capture the traffic and send it to Wireshark for analysis. As detailed by Ryan [16], the initial pairing process must be captured. The CONN_REQ packet includes the unique access address to the conversation,

Figure 2. Security Analysis Methodology Diagram

the hop interval and increment, and the CRC init. If this information is sniffed, the Ubertooth automatically follows the connection of a specified MAC address. If the collected traffic is encrypted, the attacker would likely select a new victim. However, if the data is in plain text, it provides an easy target for the attacker to see if there are any passwords or important data values available. Even if the data is obfuscated in a manner that may not be obvious at first, a determined attacker can parse through the data to interpret it.

BLE uses the client-server model where the device requesting the information, such as a phone, would be considered the client and the device providing the readings and data is the server. In the standard BLE model, when the client wishes to request new data or a value update, it sends a Read Request to the Value Attribute of a particular characteristic within the service it resides in on the BLE server [19]. The GATT on the server then interprets the request based on its UUID and sends the relevant attribute data. A handle is also assigned to a specific attribute. Wireshark parses out the data and identifies the service and characteristic UUID, as well as that attribute's handler, used in a BLE packet.

The researcher examined the Wireshark capture from the Ubertooth and parsed through the traffic between the phone and Masimo MightySat using the `btatt` filter. The overall flow consisted of ATT protocol packets where a Write Request was sent to handle `0x0018`, a Write Command to handle `0x0015`, and then Handle Value Notifications to handle `0x0017`. These packets sent the health data from the device to the phone. With this in mind, the next step in the methodology is static analysis.

LightBlue [27] was used to pull the advertised name, MAC address, manufacturer data, the model number string, and the serial number string. This information was freely advertised, and the device required no pairing to request the data. This data leakage gives vital information necessary for open search research into the device. The `HCI_snoop.log` file was pulled for inspection and filtered by the `btatt` protocol in order to see the attributes. While the Ubertooth capture did not specify who was sending or receiving a particular packet, the HCI log showed that the Sent Write Command and Sent Write Request to handles `0x0015` and `0x0018`, respectively, were sent from the phone to the device requesting an update. After this, the target device responded with similar packets labeled as Rcvd Handle Value Notification to handle `0x0017`. The values in these packets, as in the Ubertooth capture, varied in structure and were not easily parsable to determine which fields contained the pertinent data updates.

3.2. Static Analysis

The next steps in the methodology, static analysis of the Android phone log and dynamic analysis of the companion application, are both done assuming use of a rooted Android phone, the IoT device, and the corresponding mobile app. Each application on an Android phone is stored in an APK file. MobSF and `jadx` both analyze an application and decompile the .dex files in the APK into a version of the Java source code. The analyst can use these tools to identify functions of interest in the APK. For this research, the functions to identify were those that took in the BLE data to send it to be displayed in the app, used for average statistics, and storage.

The `HCI_snoop.log` file, which is enabled through the developer's options on an Android phone, captures the communication to the device through the phone's host controller interface. This log includes all communication between the phone and the device, and does not require the user to do anything other than enable the logging capability. This file captures all commands to and from the phone BLE interface and can be examined in a network analyzer such as Wireshark. Like in the first step of the methodology, this analysis in Wireshark includes the type of protocol packets (ATT, L2CAP, etc.), the opcode (read request, write request, notification, command, etc.), and the handle value that specifies the part of the characteristic

the data is coming from. Another tool useful for static analysis is LightBlue. It is an application used for testing and simulation of BLE devices. If the device is broadcasting, LightBlue can connect to the device and request advertised data.

For static analysis, both MobSF and `jadx` were used in tandem. This is because `jadx` has an easily searchable GNU user interface (GUI) that allows the analyst to search for specific strings and find where it is used in the APK. MobSF provides a breakdown of the activities, services, files, and potential security vulnerabilities present in the APK. Using both allowed correlation of the classes and functions highlighted in MobSF and `jadx` to find usage. Through static analysis of the APK, the researcher determined `BluetoothLEConnection.java` was a file of interest because it extended the `BluetoothGattCallback` class. Four main functions exist to be analyzed dynamically: `MyBluetoothGattCallback.onConnectionStateChanged()`, `MyBluetoothGattCallback.onServicesDiscovered()`, `MyBluetoothGattCallback.onDescriptorWrite()`, and `MyBluetoothGattCallback.onCharacteristicChanged()`.

3.3. Dynamic Analysis

Once functions of interest are identified statically, the next step is to pull data in real-time. `Frida` is a mobile application dynamic analysis tool that dynamically injects Javascript code into an executing process. Hooking the functions of interest allowed examination of what argument types and values into stack functions and how the application processed user data. MobSF provides standard scripts to pull standard data such as a file trace or to monitor APIs. Scripts available through MobSF were utilized, as well as custom scripts to hook specific functions for BLE data transfer.

The dynamic analysis was accomplished through use of a custom script utilizing the Frida framework. An exerpt of the code used to gather the values of interest is included in Figure 3. The researcher hooked `onCharacteristicChanged()` under `com.masimo.harrier.library.classes`'s `BluetoothLEConnection.MyBluetoothGattCallback`. `BluetoothGatt` and `BluetoothGattCharacteristic` arguments are passed into the function. Values from the characteristic are able to be pulled such as the UUID, values, descriptor, and services. The values passed into this function are in an array by default, so the researcher wrote a function to convert the array to hex values in order to better correlate with the values from the Ubertooth capture and `HCI_snoop.log`. The output resulted in several arrays starting with the value `0x77` and occasional arrays starting with `0x0f`. The arrays that began with `0x0f` showed recognizable values consistent with oxygen levels and pulse rate values.

```
var Gatt = Java.use('com.masimo.harrier.library.
    classes.BluetoothLEConnection$MyBluetooth
    GattCallback');

    Gatt.onCharacteristicChanged.implementation =
    function (arg1, arg2) {
       var valarray = arg2.getValue();
            if (valarray[0] != 119) {
       console.log("Hooked onCharacteristicChanged
    here ");
       console.log("Changed Characteristic:" + arg2
    .getUuid());
        console.log("Value:" + JSON.stringify(arg2.
    getValue()));
       console.log(toHexString((arg2.getValue())));
       console.log("getDescriptor: " + JSON.
    stringify(arg2.getDescriptors()));
       console.log("Service:" + arg2.getService());
        this.onCharacteristicChanged(arg1, arg2);}
```

Figure 3. Hooking Code Example

Filtering the script to only show the arrays beginning with `0x0f` repeatedly showed that the eighth and tenth bytes were updated with the oxygen and pulse rate readings from the device. The hex value of the array resulted in a value with the content of `0f05000000000063004a100a005f00f5`, where the values `63` and `4a` represent the oxygen level of 99 and pulse rate of 74, consistent with the data shown on both the device and in the GUI.

4. Results and Discussion

Upon completing the methodology outlined, the analyst was able to determine the device had data leakage. Table 3 outlines the relation between the Ubertooth capture, the `HCI_snoop.log` on the phone, and the data gleaned from `Frida` hooking into the vendor application. Once the packet structure was determined using `Frida`, the HCI and Ubertooth pcap files were examined to look for a similar composition. Since the manufacturer did not employ encryption, the security examiner was able to determine the location of the bytes that correspond to the health statistics in the network packet. The `HCI_snoop.log` and values from the application are identical for the first 13 bytes. The significance for the last bytes are currently undetermined, but would not be required to know this in order to sniff the traffic and find the health statistics of interest. The Ubertooth capture traffic value was the same as the HCI log, save with `0x77` at the

Table 4. Structure of Health Statistics

	BluetoothLEConnection Function Hooked	HCI_snoop.log	Ubertooth Sniffed Traffic
Info Field	Value of BluetoothGattCharacteristic	Rcvd Handle Value Notification	UnknownDirection Handle Value Notification
Handle	N/A	0x0017	0x0017
UUID	54c210022a72004b4f11e49fe20002a5d5	54c210022a72004b4f11e49fe20002a5d5	54c210022a72004b4f11e49fe20002a5d5
Value	0f05000000000063004c14ff206100ad	0f05000000000063004c14ff206000b8	770f05000000000063004c14ff206000b8
Packet Length	N/A	28 bytes	43 bytes

beginning. Furthermore, the packet size for the HCI log was predictable at 28 bytes for each packet that included the information, and the sniffed traffic had a length of 43 bytes. Knowing this information would allow the attacker to easily filter the traffic to only see packets of interest.

5. Future Work

There are many opportunities to build upon this research. For example, it would be beneficial to determine if the communication between the phone and the device is susceptible to injection attacks allowing the data to be manipulated in real time. This could be done using the Ubertooth hardware and corresponding command-line tool to see if data could be replayed to the application, and if other values could be input instead. This would require better understanding of the entirety of packets sent, rather than necessarily just the packet with the information of interest. `Frida` injection could also determine vulnerabilities to examine if the application and companion app are vulnerable to fuzzing or what values it will accept without raising alarm.

Reverse engineering the hardware is also a viable direction for research. The researcher was unable to do so in this experiment due to having to maintain the functionality of the device. However, the hardware could contain pertinent data. A mechanism to force an over the air update to the device could be determined, such as with Gutierrez [19], to permanently alter the data or brick the system altogether.

Additionally, it would be beneficial to examine the Masimo MightySat's competitor, the Nonin Onyx II 9560 [32]. The researcher could use the same methodology outlined in this paper to determine if the Nonin device also has data leakage. Future researchers should also explore other like medical devices with BLE to compare the security features of each.

Reporting false measurements on a medical device could result in detrimental physical effects on the patient, especially if medicine is prescribed based on the data, where the user could get the incorrect dosage or no dosage at all. These future research areas could help secure medical devices from other security attacks.

6. Conclusion

Correlation of the data gathered through passive sniffing, static analysis, and dynamic analysis allowed the researcher to determine the packet structure containing the oxygen and pulse rate values. In Table 3, the oxygen value and pulse rate are the eighth and tenth bytes for `BluetoothLEConnection` hooking and for the `HCI_snoop.log`, and the Ubertooth sniffed traffic has those values in the ninth and eleventh bytes. While the manufacturer of the device may have intended to secure the device communication with a proprietary format, the researcher successfully determined the structure of the packets containing medical data.

Acknowledgement

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

References

[1] G. Maayan, "The IoT Rundown For 2020: Stats, Risks, and Solutions." https://securitytoday.com/articles/2020/01/13/the-iot-rundown-for-2020.aspx.

[2] G. A. Office, "Internet of Things: Enhanced Assessments and Guidance Are Needed to Address Security Risks in DoD." https://www.gao.gov/assets/690/686203.pdf/.

[3] K. Foote, "A Brief History of the Internet of Things." https://www.dataversity.net/brief-history-internet-things/.

[4] A. Gibbai, "Kevin Ashton Describes the Internet of Things." https://www.smithsonianmag.com/innovation/kevin-ashton-describes-the-iot-180974.

[5] A. Shahzad, Y.-G. Kim, and A. Elgamoudi, "Secure iot platform for industrial control systems," in *2017 International Conference on Platform Technology and Service (PlatCon)*, pp. 1–6, IEEE, 2017.

[6] M. Aleksandrova, "IoT in the Workplace: Smart Office Applications for Better Productivity." https://www.iotforall.com/iot-smart-office-applications/.

[7] M. Ryu, J. Yun, T. Miao, I.-Y. Ahn, S.-C. Choi, and J. Kim, "Design and implementation of a connected farm for smart farming system," in *2015 IEEE SENSORS*, pp. 1–4, IEEE, 2015.

[8] Econsultancy, "10 Examples of the Internet of Things in Healthcare." https://econsultancy.com/internet-of-things-healthcare/.

[9] M. Hossain, R. Hasan, and A. Skjellum, "Securing the Internet of Things: A Meta-Study of Challenges, Approaches, and Open Problems," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 220–225, IEEE, 2017.

[10] Y. B. Zikria, S. W. Kim, O. Hahm, M. K. Afzal, and M. Y. Aalsalem, "Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution," 2019.

[11] AvSystem, "IoT Standards and Protocols Guide: Protocols of the Internet of Things." https://www.avsystem.com/blog/iot-protocols-and-standards/.

[12] Masimo, "MightySat Rx Fingertip Pulse Oximeter." https://www.masimo.com/mightysatrx/.

[13] AndroidDeveloper, "Bluetooth Low Energy Overview." https://developer.android.com/guide/topics/connectivity/bluetooth-le.

[14] BluetoothSIG, "Gatt Services." https://www.bluetooth.com/specifications/gatt/services/.

[15] P. Rentz, "OWASP Releases Latest Top 10 IoT Vulnerabilities."

[16] M. Ryan, "Bluetooth: With low energy comes low security," in *7th {USENIX} Workshop on Offensive Technologies ({WOOT} 13)*, 2013.

[17] "Ubertooth." https://github.com/greatscottgadgets/ubertooth/.

[18] "Wireshark." https://wireshark.org.

[19] J. G. del Arroyo, "How Do I BLE Hacking." https://www.youtube.com/watch?v=oP6sx2cObrY.

[20] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pp. 129–142, IEEE, 2008.

[21] M. Rahman, B. Carbunar, and M. Banik, "Fit and vulnerable: Attacks and defenses for a health monitoring device," *arXiv preprint arXiv:1304.5672*, 2013.

[22] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or foe? your wearable devices reveal your personal pin," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 189–200, 2016.

[23] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing.," in *NDSS*, 2018.

[24] "Fuzzing." https://owasp.org/www-community/Fuzzing.

[25] G. Campbell, "What is Taint Analysis." https://dzone.com/articles/what-is-taint-analysis.

[26] J. Hedberg, "Bluez." http://www.bluez.org/.

[27] PunchThrough, "Lightblue." https://punchthrough.com/testing-bluetooth-low-energy-devices/.

[28] Abraham, "Mobile Security Framework." https://github.com/MobSF/Mobile-Security-Framework-MobSF.

[29] Skylot, "Jadx." https://github.com/skylot/jadx.

[30] NowSecure, "Frida." https://www.frida.re/docs/home/.

[31] A. Developer, "Android Debug Bridge." https://developer.android.com/studio/command-line/adb.

[32] "Massimo MightySat Brochure." https://www.masimo.com/siteassets/us/documents/pdf/plm-11294e_brochure_mightysat_rx_us.pdf.