Blue's Clues:
Practical Discovery of Non-Discoverable Bluetooth Devices

Tyler Tucker, Hunter Searle, Kevin Butler, Patrick Traynor
Florida Institute for Cybersecurity Research (FICS)
{tylertucker1, huntersearle, butler, traynor}@ufl.edu

***Abstract*—Bluetooth is overwhelmingly the protocol of choice for personal area networking, and the Bluetooth Classic standard has been in continuous use for over 20 years. Bluetooth devices make themselves *Discoverable* to communicate, but best practice to protect privacy is to ensure that devices remain in *Non-Discoverable* mode. This paper demonstrates the futility of protecting devices by making them Non-Discoverable. We introduce the *Blue's Clues* attack, which presents the first direct, non-disruptive approach to fully extracting the permanent, unique Bluetooth MAC identifier from targeted devices in Non-Discoverable mode. We also demonstrate that we can fully characterize device capabilities and retrieve identifiers, some of which we discover often contain identifying information about the device owner. We demonstrate Blue's Clues using a software-defined radio and mounting the attack over the air against both our own devices and, with institutional approval, throughout a public building. We find that a wide variety of Bluetooth devices can be uniquely identified in less than 10 seconds on average, with affected devices ranging from smartphones and headphones to gas pump skimmers and nanny-cams, spanning all versions of the Bluetooth Classic standard. While we provide potential mitigation against attacks, Blue's Clues forces a reassessment of over 20 years of best practices for protecting devices against discovery.**

I. Introduction

Bluetooth is continually the worldwide standard for local wireless networks [1]. Connectivity between devices has gone far beyond simply the replacement of wires, where virtually every device that supports wireless communication also enables Bluetooth functionality. One of the reasons for Bluetooth's ubiquity is the ease by which connections with other devices can be enabled. The *Discovery* process involves a device broadcasting its presence to other devices within radio range while scanning for other devices that are similarly advertising their capabilities. This discovery process serves as a precursor to Bluetooth devices pairing, and subsequently communicating, with each other.

From its inception, Bluetooth has considered security and privacy concerns. The original Bluetooth 1.0 specification specified that Bluetooth devices should support *Non-Discoverable* mode, whereby devices (called "silent devices" in the Bluetooth specification) do not respond to inquiries from other devices [2]. By preventing discovery, Bluetooth devices can prevent identifiable and unique information about them from being made accessible. Current best practices relating to Bluetooth security recognize that devices in Discoverable mode are subject to attack and hence, recommendations from organizations such as NIST and CISA recommend keeping Bluetooth devices in Non-Discoverable mode [3], [4].

While there has been substantial work examining security aspects of the Bluetooth Classic protocol over the past 20 years [5]–[27], only a small number of papers have addressed Non-Discoverable mode [5]–[10]. Early work was solely theoretical, calling for up to 80 transmitting radios and between 24 hours and a week to brute force the Bluetooth address range to recover the Bluetooth address [5]. Recent work substantially advances Bluetooth device identification, bringing identification time down to seconds, but can only provide probabilistic guarantees about whether the device is one that has been previously seen and does not identify the device characteristics [6].

In this paper, we demonstrate the futility of using Non-Discoverable mode as a means of assuring the privacy of Bluetooth Classic devices. Our Blue's Clues attack goes well beyond past work by incorporating active communication with a target device using the Bluetooth Link Manager Protocol (LMP) to extract identifiable information. Moreover, we demonstrate that the full Bluetooth address, which is unique and permanent to a device, can be directly exfiltrated from a device in Non-Discoverable mode. Such an attack occurs transparently to the user and can be completed without the target device ever raising any alert or notification about the communication.

The primary contributions of this work are as follows:

- **Develop New Active Attack Breaking Bluetooth Non-Discoverable Mode:** Prior work has demonstrated the possibility of using SDRs to recover Bluetooth device address candidates. Our active Blue's Clues attack goes further, and demonstrates the ability to not only recover all device information available in Non-Discoverable mode, but also extract the *unique and permanent* device identifier for any Bluetooth device. Moreover, we demonstrate that this information can be used to launch pre-authentication attacks.
- **Find and Identify a Wide Range of Devices:** Bluetooth radios can be found on a wide range of devices, whose purposes can range from benign entertainment (e.g., wireless earphones, smartphones) to more malicious intents (e.g., gas pump skimmers, hidden cameras). We collect 21 different devices spanning Bluetooth versions 2.1 to 5.2 and demonstrate that our attack is capable of finding and identifying all of them. We then demonstrate that our

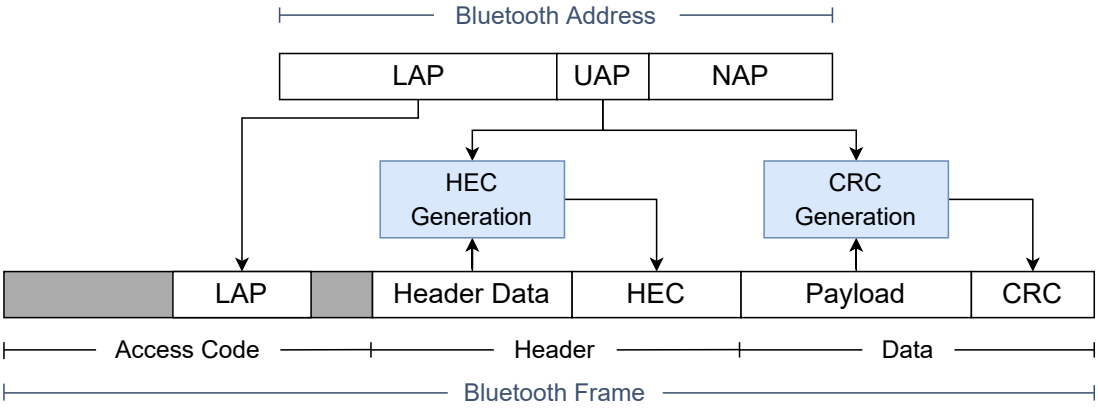

Fig. 1. This diagram highlights the relationship between the Bluetooth MAC address and the Bluetooth packet frame. The Lower Address Portion (LAP) is inserted directly into the Access Code portion of the frame while the Upper Address Portion (UAP) is combined with the Header Data and Payload to create the HEC and CRC codes, respectively. The Non-Significant Address Portion (NAP) does not contribute to any portion of the packet.

approach can find far more devices than the traditional discovery process [21]–[23] through an IRB-approved study of our office space. Regardless of the age of the device or the Bluetooth specification that it supports, every device we test is vulnerable to Blue's Clues.

- **Present Potential Mitigations:** Bluetooth-capable devices run the gamut of capabilities and interfaces, making one simple mitigation unlikely to completely address our attack. As such, in addition to disclosing a protocol fix to the Bluetooth SIG, we discuss a number of point solutions for constrained devices (i.e., those without interfaces to make authentication easy) and evaluate one mitigation option on a Bluetooth device.

The Blue's Clues attack forces a reassessment of over 20 years of best practices for Bluetooth security and with over 4 billion Bluetooth devices shipped in 2020 and billions more deployed devices [1], significant efforts will need to be made to address these issues in design and implementation of Bluetooth.

The remainder of this paper is organized as follows: Section II provides background on Bluetooth; Section III discusses our threat model; Section IV details the methodology of the Blue's Clues attack; Section V discusses the implementation of our attack; Section VI evaluates the success of our attack against a wide range of Bluetooth-enabled devices and peripherals; Section VII discusses related issues and potential mitigations; Section VIII contextualizes this effort against related work; and Section IX provides concluding remarks.

II. Background

A. Overview

Bluetooth is a wireless personal area network protocol operating in the 2.4 GHz ISM frequency band. Its spectrum is divided into 79 physical channels spaced 1 MHz apart. Only one channel is used by each connection at a time. Communicating devices use adaptive frequency hopping, switching between a subset of the Bluetooth channels 1600 times per second. The subset of channels is negotiated between devices in response to interference or noise in certain channels. In order to establish a connection, two devices must exchange addresses, as well as synchronize frequency hopping sequences. Connections are initiated by a central device, although central and peripheral roles[1] may be switched later in the process. The process of establishing a connection between unfamiliar devices is called pairing and is composed of two parts, *inquiry* and *paging*.

B. Bluetooth Pairing

The central device will begin the pairing process by transmitting an inquiry packet on one of three designated inquiry channels. This packet uses a general inquiry address, so it is not directed to any particular device. Bluetooth devices in Discoverable mode monitor these channels, waiting for inquiry packets. When they receive the packet, they will respond with an inquiry response message containing their full address, device capabilities, and name. The initiating device then has enough information to make a full connection with the responding device and will begin the paging process to establish a direct connection.

The paging process is similar to the inquiry process, except the peripheral device's address is used in place of the general value used for inquiry. This assures that only the intended device will respond. As the hopping patterns have yet to be synchronized, the central device has a low chance of transmitting on the correct channel. To overcome this, the central device will send many paging request messages in quick succession until it receives a response. After the response is received by the central device, it will send a frequency hopping synchronization (FHS) packet to provide the peripheral with the information it needs to synchronize its frequency hopping scheme with that of the central device. Once the peripheral

[1]The Bluetooth specification uses the terms *Master* and *Slave*, but we choose to replace these terms throughout our paper in light of community discussion [28]–[30].

responds to the FHS packet, the connection is established and the two devices can begin exchanging information.

In the Non-Discoverable mode, Bluetooth devices do not respond to inquiry messages. This prevents the creation of any connections with unknown devices. However, *regardless of discoverability, the device will always respond to paging packets formed with its address*. Thus, the only requirement for establishing a connection with an unknown device is to know the necessary portions of that device's address.

C. Bluetooth Addressing

There are three parts to a Bluetooth address: a 24-bit Lower Address Portion (LAP), an 8-bit Upper Address Portion (UAP), and a 16-bit Non-Significant Address Portion (NAP) as shown at the top of Figure 1. Most Bluetooth packets are categorized as ID packets, meaning they contain the LAP within the packet. Additionally, the UAP is used as a parameter to a linear feedback shift register in generating the cyclic redundancy check (CRC) and header-error-check (HEC) sections of a packet's header, shown at the bottom of Figure 1. The NAP is not used in producing any other values, meaning that it cannot be recovered by inspecting any packet headers.

Aside from inquiry packets, which use a reserved UAP and LAP, packets are formed from a device's UAP and LAP so that only the intended device will receive and respond to a packet. These come from the peripheral's address during the paging process, and the central device's address once the connection has been established. The header containing the LAP, CRC, and HEC is whitened, or encoded, before transmission, to prevent long sequences of bits from being transmitted (e.g., 00000000 or 11111111). However, the header can be de-whitened with relative ease [6], [7]. Thus, the LAP is readily available in every packet. The UAP is not directly present in the packet, but the CRC and HEC contain enough information to narrow down the possibilities for the UAP. By sniffing a packet with a HEC or CRC, the packet data, generation algorithm, and result (shown in Figure 1) are known, and so the UAP can be recovered using this information from several packets as shown by prior work [6], [7]. Both the LAP and UAP are necessary to establish a connection.

D. Link Manager Protocol

Bluetooth Classic uses the Link Manager Protocol (LMP) to control all aspects of the Bluetooth connection between two devices [31]. This includes the negotiation and management of logical transports and logical links, as well as physical links. LMP connects the Link Managers (LM) between the devices, which sit just above the Link Control (LC) level that decodes packets received from the physical layer. Messages are exchanged in connected sets, called transactions, with each attempting to achieve a particular purpose. All LMP messages strictly apply to LM, LC, or physical layers. These layers are situated in the Bluetooth controller, and no messages are passed directly to the host device.

LMP begins immediately after the paging process. The paging device uses LMP messages to request device information such as clock offset, Bluetooth version, device capabilities, and device name. Using this information, the paging device requests a connection and, if accepted, negotiates pairing, authentication, and encryption. LMP is also responsible for determining the roles of the devices in a connection, and so allows these roles to be changed via an *LMP Role Switch* message. This is useful when a device designed to take the peripheral role is the one to establish the connection (e.g., turning on a pair of wireless headphones that automatically connects to your phone). The device that creates the connection is set as the central device by default, so the headphones would need to perform a role switch with the phone to become the peripheral device. The host, and by extension the user, is not aware of any LMP message exchange. If the process is terminated, either by a device not responding, or a device sending an LMP detach message, the host is never notified of the messages that were sent.

III. Threat Model

In this paper, we assume that an adversary can capture all communications within the Bluetooth spectrum and transmit legitimate Bluetooth packets with arbitrary addresses, access codes, and payloads. These are both possible using commercial off-the-shelf devices and open-source code, meaning that a relatively unsophisticated adversary is capable of performing this attack. Additionally, the attacker requires no prior knowledge of any devices within the attack area, and only minimal time in proximity to a device to complete the attack.

The threat model in this paper is fundamentally different from the threat model assumed when the Non-Discoverable mode was originally introduced. At that time, the possibility of full-band eavesdropping on commodity hardware was nonexistent. Only with the recent rise in the availability of low-cost software-defined radios have attacks of the kind we present become feasible. Additionally, Bluetooth-compliant transceivers with open-source code, such as the Ubertooth, have made Bluetooth research more accessible to a wider swath of the population than ever before. This lower barrier to entry into Bluetooth has opened the path for more sophisticated attacks that would not have been possible 20 years ago.

IV. Methodology

There are two main challenges with finding Non-Discoverable devices. First, they will not respond to any broadcast messages, so no attempts to "ping" them will yield results if the address is not previously known. Second, we cannot recover the entire Bluetooth MAC address by passively capturing generic Bluetooth packets, as they only reveal a portion of this address. Therefore, a passive eavesdropping attack can, at best, recover the lower portions of the MAC address in the UAP and LAP. This provides a pseudo-unique identifier[2] for a local device but does not provide any further

[2]We use the term *pseudo-unique* here because the MAC address fragment is not guaranteed to be unique without the upper NAP portion of the address, although the probability of two local devices having the same lower address portions is low.

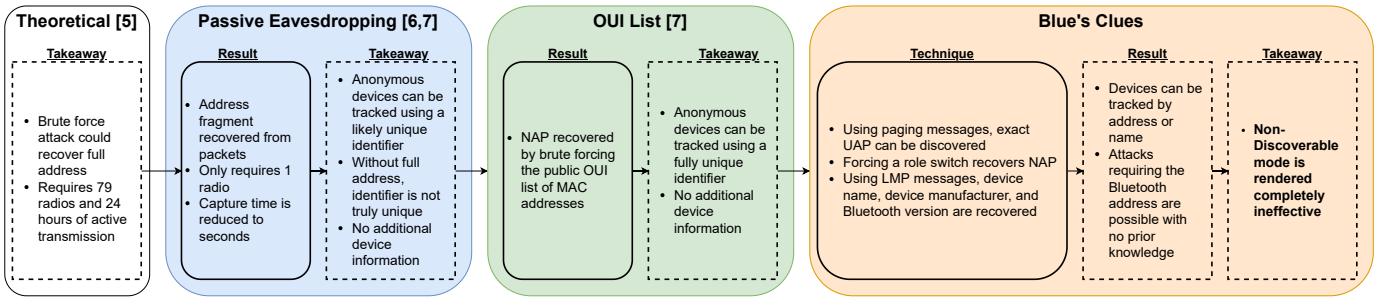

Fig. 2. We compare Blue's Clues with past work to show our improvements over theoretical and practical solutions to detecting Non-Discoverable devices. We are the only attack to recover all information designed to be concealed by the Non-Discoverable mode, as past solutions stop once they recover at most the device's MAC address.

information. Our contribution is an attack that combines both passive eavesdropping and active communication to overcome these challenges. With this attack, we can find all communicating devices, recover the entire Bluetooth MAC address, *and* retrieve the same information from Non-Discoverable devices as the standard inquiry process does from Discoverable devices, thereby **fully defeating the Non-Discoverable security feature of Bluetooth Classic**.

A. Motivation

With the increased adoption of Bluetooth in consumer devices since the inception of the Non-Discoverable mode [32], past approaches to location tracking based on the inquiry process cannot find most modern devices. We, therefore, aim to show that not only is location tracking still possible, it is practical under a new threat model. This new model no longer assumes devices are Discoverable but considers any device using Bluetooth vulnerable. While other works have aimed to passively detect these Non-Discoverable devices, they cannot (1) recover a full MAC Bluetooth address, (2) gather any further information on the device, nor (3) interact with the device, all of which our attack overcomes. Due to these limitations, passive attacks [6], [7], [24], [25] cannot claim to break the Non-Discoverable mode of Bluetooth, as they do not recover all of the information that this security feature is designed to hide. We highlight this distinction in Figure 2.

1) **Inability to directly recover full Bluetooth address** This first limitation of a passive attack is important because the full MAC address of a Bluetooth device will never change. The address, therefore, acts as a unique and permanent identifier that can be associated with the user. With this, an adversary can ping for a recovered MAC address anywhere and will always receive a response if the device is within range. This allows an adversary to check for user presence in any location and guarantees a response as long as Bluetooth is turned on. While one past work [7] has proposed an active portion of their attack to iteratively guess at NAPs until one is confirmed using a public OUI list of MAC addresses [33], our solution removes the guesswork and can directly extract the NAP.
2) **No device information accessible** The second current limitation is that no further details about the device are recovered through passive attacks. With our attack, we can recover device owner names (e.g., *John's iPhone* or *Kevin's Pixel*), which are personally identifying *on top of* the unique and permanent MAC address. Additionally, we gather device manufacturer and device types, all of which can be leveraged to filter for an individual (e.g., the adversary knows their target is using an iPhone, so they search for smartphones made by Apple).
3) **Lack of interaction** The third limitation is that, by definition, passive attacks do not interact with any device. By establishing a connection with each device, we create a vector for exploiting vulnerabilities that propagate through Bluetooth. The information we gather allows us to reason about the device model, allowing us to target a vulnerability from a subset of devices (e.g., Google Pixels). This in turn can allow for a DoS attack in a local area or even remote code execution against certain devices. We show that this concern is a reality by executing a known exploit against an Android smartphone after discovering it automatically through our attack. Additionally, we can extend our active communication to complete device pairing with devices that do not require a Bluetooth PIN to accept a connection. The majority of Bluetooth accessories (e.g., headphones and speakers) do not require PINs to accept a connection and therefore will freely pair with any device that knows its MAC address.

In short, our attack fully breaks Non-Discoverable mode and allows for an attacker to recover sensitive information and potentially connect directly to all Bluetooth devices that are turned on.

B. Passive Eavesdropping

A purely passive attack will recover portions of MAC addresses but fail to gather any further details about local devices. Instead, it only provides possibilities for a section of the MAC address of a device. We can, however, use these recovered MAC address fragments (consisting of the LAP and UAP) to establish our own connection to these devices to fill this gap. Therefore, our attack begins by passively eavesdropping on a

subset of channels of the Bluetooth spectrum to recover LAPs of nearby devices. The exact number of channels affects the speed at which devices will be discovered but is otherwise irrelevant to the process.

We employ the current state-of-the-art in passive Bluetooth monitoring [6] as a method for recovering addresses from local traffic. This program is open-source [34] and only requires a software-defined radio to operate. While monitoring a set of channels, all packets are captured and the header is *de-whitened* (decoded) to reveal the LAP of the central device of the Bluetooth connection. Recall from our background section that Bluetooth whitening is simply a method of encoding to prevent sequences of repeated bits (e.g., 00000000) from being transmitted for accuracy improvement in demodulation [35]. Next, the program recovers error-checking codes *HEC* and *CRC*, which help narrow down the possibilities for the UAP of the device. The UAP, along with the 8-bit central device clock, is used as a seed when generating these values, so the code can generate HEC values for a packet using combinations of these two values and compare that to the HEC that was recovered over the air. When the two HECs match, the UAP is logged as a viable candidate. This method provides two possible UAPs to consider in the best case, which is a limitation of the SDR approach we adopt. With two possible UAPs, we have two options for MAC address fragments for a device but have no further information. Our work builds on this process, so our contribution begins *after* this step.

While the existence of a device is revealed this way, this is not equivalent to the information provided by the inquiry process against Discoverable devices and therefore does not fully defeat the Non-Discoverable mode. We can, however, use these address fragments to add a second stage to our discovery process in which we establish a short connection with the central devices to fill this gap.

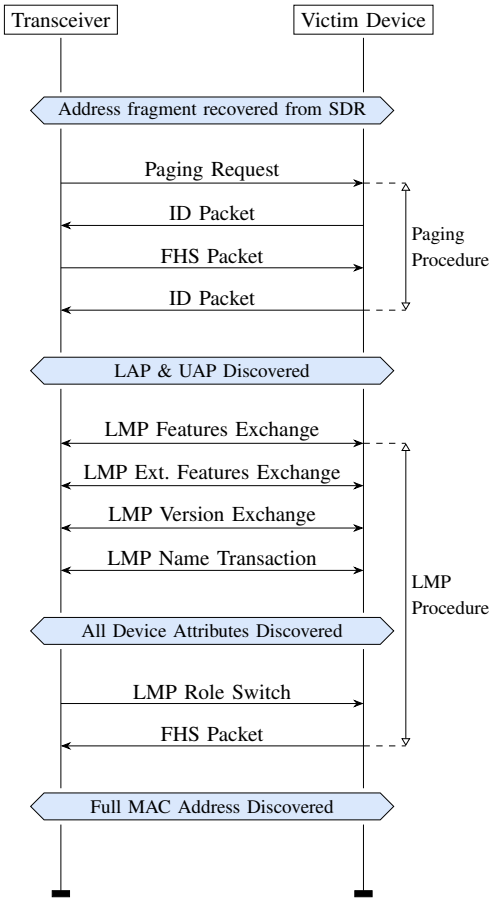

Fig. 3. Sequence Diagram of the *Blue's Clues* attack. Once a valid *LAP+UAP* MAC address fragment is produced by the SDR program, the transceiver will perform a standard paging procedure. If a second ID packet is received, the transceiver can then send LMP messages to the foreign device to gather information on it.

C. Active Communication

To overcome Non-Discoverable mode, we need to recover the information it conceals past only a portion of the MAC address. **We achieve this by establishing a short connection to the device during which we exchange information between our device and the target device in our active stage, which is our main contribution and the link to overcoming Non-Discoverable mode**. When we enter the active stage, we have a device's LAP and a small number of possible UAPs. Figure 3 contains a sequence diagram overview of our active connection process. In the first portion of our active process, we iteratively attempt to establish a connection to the device using the standard paging process outlined in our background. We exhaustively test the remaining possibilities by creating paging requests with the known LAP and a possible UAP (this is labeled as *Paging Procedure* in our diagram). We transmit these test packets on many different channels to increase the likelihood that the device will receive at least one, producing hundreds of attempts per second. If we receive a response from the device that ends the paging process, we can conclude that our UAP is correct and we can immediately begin the second portion of our active process.

Our second portion, called *LMP Procedure* in Figure 3, consists of transactions of LMP messages between our transceiver and the foreign device. These messages are all sent before authentication occurs, so all devices will respond to this process even though our MAC address is completely unknown to them. The *Features* and *Extended Features* LMP messages provide the capabilities of the device (e.g., encryption, Bluetooth LE, Secure Simple Pairing). Next, the *Version* message contains what Bluetooth version the device uses and the manufacturer of the Bluetooth module. The *Name* message will tell us what human-readable name the device is given. We noted this information in our motivation section because it is particularly impactful on location privacy, as the name of the device often includes the name of the owner, such as *Steve's iPhone*. These four messages are sent whenever two devices begin pairing, so they mirror the behavior of the standard pairing procedure.

No LMP message exists to retrieve the full Bluetooth address, but we can utilize the *LMP Role Switch* message to recover it directly. Recall that a role switch is used when a

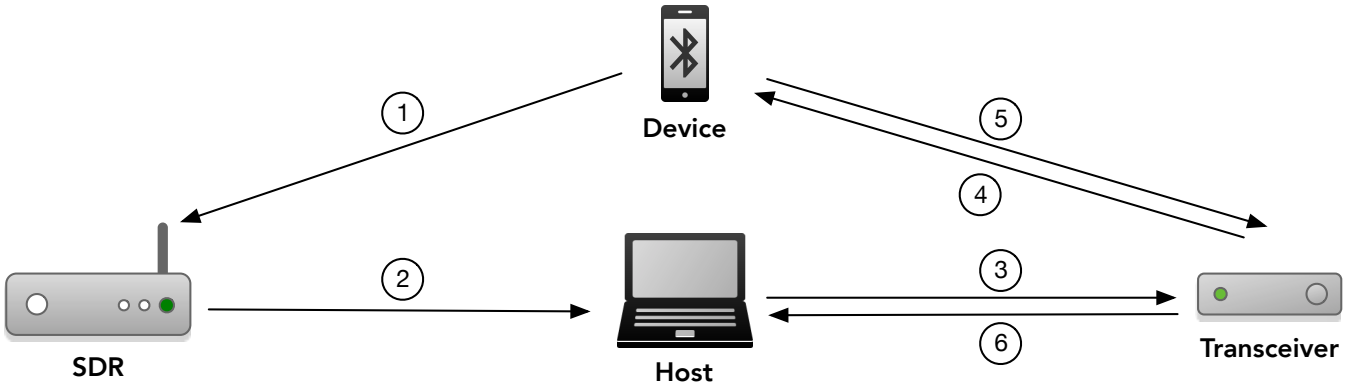

Fig. 4. Blue's Clues Attack Architecture, consisting of six stages. The attack first (1) detects a transmission from a nearby Bluetooth device and (2) sends the data to a program running on a computer. The address fragment for the Bluetooth device is then (3) forwarded to the transceiver which will (4) establish its own connection to the device and (5) receive information from the device. Finally, the transceiver will (6) send the gathered information back to the host program for database logging.

device would like to change roles from either a central to a peripheral device or a peripheral to a central device. When a role switch message is accepted by a device, that device immediately sends over a frequency hopping synchronization (FHS) packet, similar to the one we send out during the paging procedure portion of our connection. This message contains all three portions of the Bluetooth MAC address as well as both the *Major Device Class* and *Minor Device Class* fields (e.g., *Phone* and *Smart Phone* for an iPhone). Therefore, we can use the role switch procedure in an unintended manner as a method of recovering the device type and the NAP portion of the MAC address. The only aspect of our attack that deviates from standard pairing behavior is our immediate disconnection after performing the role switch. The end-user will not be aware of this behavior, however, because this messaging procedure is not translated to higher layers in the Bluetooth stack.

While the NAP is not used as a seed for generating anything about the connection like the other portions, it does tell us what the OUI (manufacturer) of the host device is (e.g., Google Inc. or Apple). Also, the retrieval of the NAP completes the entire Bluetooth MAC address. Since this value is static for Bluetooth Classic, this serves as a *permanent and unique* identifier for the device.

Once we retrieve the complete set of identifying information, we terminate the connection before any communication is sent to the host. The host, and therefore the user, is never aware of our presence, nor is there any record that we have communicated with the device.

V. Implementation

Our goal is to automatically detect Bluetooth transmissions and connect to the corresponding transmitters, so our attack consists of two stages. The first stage requires a radio receiver to monitor a subset of Bluetooth channels for ID packets which can be *de-whitened* to reveal the LAPs of the devices that transmitted them. Once the LAP is recovered, this stage must then begin recovering the UAP of that device. Both steps are crucial because both address portions are required for connection establishment. Previous work in Bluetooth sniffing [6], [7] can fulfill this role.

The second stage requires a Bluetooth transceiver to perform a paging procedure for every possible address it receives from the radio until it confirms that it knows the valid address for the device. We desire two separate radios here so that we can simultaneously listen for Bluetooth traffic and attempt a direct connection to a local device. Once the address is known, the transceiver must establish a connection with the device to exchange LMP messages before any authentication is challenged. Once we have collected all of this information, we can simply stop communicating with the foreign device. The end-user of the device will receive no notification that any communication took place, as LMP messages are not propagated to higher layers.

Our setup consists of three components: an SDR, a Bluetooth transceiver, and a host machine. Figure 4 provides a diagram of the implementation with embedded dataflow. Packets are first gathered by our SDR and sent to the host computer for de-whitening and UAP computation in Steps 1 and 2. Then, the recovered LAP along with a list of possible UAPs is sent to the transceiver host program which selects one MAC address fragment to send to the transceiver in Step 3. The transceiver will page that address fragment and establish a connection with the corresponding device in Step 4, assuming the address is valid. If the address is invalid, the transceiver notifies the host program to send another option. During a connection, the transceiver will send out LMP request messages and send the responses to the host for logging in Steps 5 and 6. These LMP messages are listed in Figure 3 as *LMP Procedure*. Finally, the transceiver will disconnect from the foreign device by simply ceasing to send any further messages and waiting for the next address to try to connect to.

A. Hardware Implementation

We chose to use the Ettus Research USRP B210 SDR [36] running open-source code [34] to simultaneously monitor 8 channels of the Bluetooth spectrum for local traffic. The code both recovers LAPs from Bluetooth packets and iteratively narrows down the UAP of a nearby Bluetooth device to two possibilities. This program, therefore, fits the requirements of our attack, as we require both address portions to establish a connection with an external device. Additionally, this code allows the user to expand to two B210 SDRs to cover the entire Bluetooth spectrum. We decided to use a single SDR to (1) keep the overall cost of the setup to a minimum and (2) make the setup as portable as we could. While other solutions for Bluetooth sniffing exist [37], they rely on monitoring a single frequency at a time. This method misses out on significantly more packets and therefore would require much longer exposure to a connection to accurately recover the device UAP.

Our chosen Bluetooth transceiver is an Ubertooth One [38], a popular and inexpensive embedded Bluetooth prototyping platform that connects to a host through USB. This portable solution gave us access to lower Bluetooth layers, allowing us to send arbitrary LMP messages and pass the responses to a host program. Open-source code is readily available [39] and supports paging, inquiry, and LMP procedures, making it ideal to implement our attack.

While the Ubertooth itself can act as a Bluetooth sniffer, we chose to use an external SDR for two reasons. First, like other solutions, the Ubertooth can only monitor a single channel at a time and will therefore be inefficient at discovering and narrowing down addresses. Second, the Ubertooth cannot monitor for addresses and transmit simultaneously, so our setup would miss out on even more packets while attempting a connection through paging. We employ a laptop running Ubuntu 18.04 LTS as our host machine (both natively and through a VM) but note that any device supporting the Ettus USRP [40] and Ubertooth [39] libraries will be able to run our attack. Our physical hardware setup is shown in Figure 5.

B. Software Implementation

Both the Ubertooth and the SDR have their own host programs, so we use ZMQ sockets for inter-process communication. Whenever the sniffer program discovers a new device, it will send both the LAP and a list of possible UAPs to the transceiver code. Our goal is to discover devices as quickly as possible, so we do not wait until the SDR program has completely narrowed down the UAP to begin attempting a connection with the device. We instead start attempting a connection to a given address once the list of possible UAPs contains four or fewer options.[3] Additionally, our code immediately stops attempting a connection to an address that is ruled out by the SDR program.

[3]We determined this value by testing several options until we found the one that gave us the best overall performance.

Fig. 5. Interior shot of the Faraday cage we used during our private tests. Our setup and some assorted Bluetooth devices are shown within the cage.

In the worst case, the SDR program will provide 32 possible UAPs for the Ubertooth to try. However, the SDR program typically reduces that down to 2-4 possibilities within 2 seconds. As soon as the transceiver program receives a list from the SDR program, it will begin transmitting MAC address fragments through paging messages until it receives a response to its FHS packet. If no explicit response is received, we can reasonably conclude that our UAP guess is false and move on to the next possibility. We implemented a timeout feature of four seconds to prevent the Ubertooth from spending too much time waiting on a response to a paging request message. We determined this value by measuring how long an average paging procedure took at around three seconds and set our value one second higher as a buffer.

The transceiver code uses a priority queue to determine the order of devices to establish a connection to. The devices with the smallest amount of possible UAPs are handled first, as the Ubertooth should require less time with those on average. This also gives the SDR code time to further narrow down the UAP of the other devices, which will improve the transceiver performance when the device LAP moves to the front of the queue. When the transceiver successfully discovers a device, it logs the total time taken to a separate file that we use to evaluate the performance of the attack. Finally, we log all device information including MAC address, device name, chipset manufacturer, and Bluetooth version into an SQLite database and remove the LAP of the device from the priority queue to move on to other devices. Our code is available at https://github.com/TylerTucker/BluesClues.

VI. Evaluation

A. Ethics Statement

All studies involving identifying Bluetooth devices carry with them inherent privacy risks. By definition, these studies cannot target specific devices, but retrieve information from all nearby devices to be able to identify them. Any experiments

run outside of a carefully controlled lab environment will collect information from potentially uninformed and unwilling participants. This privacy risk represents a significant ethical challenge. Some recent works have chosen not to address the ethics of their research, either by labeling the information "publicly available" or by ignoring the question altogether. We view this approach as dangerous and irresponsible.

We decided to address the ethical question directly in the design of our research by separating our studies into two pieces. In designing and testing the functionality of our setup, we exclusively tested our own devices, eliminating the privacy risk to others. To explore the implications of this attack, we needed to run a study in a more realistic scenario, which meant we would need to test on devices outside our control. Before executing this test, we sought approval from our Institutional Review Board (IRB), Office of Security and Privacy, and our sponsor's Human Research Protection Official (HRPO). With their guidance, we designed a study that would be limited to a single building owned by our university, and we notified all students and staff who could potentially be in the building on the days that we tested. Additionally, we gave all students and staff the opportunity to have their devices removed from our database and all information we collected on them deleted. No information collected during this study has nor will ever be shared with anyone outside of the authors of this work. It is crucial for our work, as well as future work, to consider the privacy risk to individuals who might be identified by experiments and mitigate this risk to the greatest possible degree.

We recognize that a larger study in a public place may have discovered many more devices that were not in our test set; however, we believed the potential consequences to uninformed users to be too significant to warrant such an action.

B. Goals

We formulate the purpose of our experiments into several research questions:

- ***RQ1*** Can our attack gather information from a device within seconds of detection?
- ***RQ2*** Does our attack apply to a variety of Bluetooth devices?
- ***RQ3*** Can our attack perform efficiently in the presence of significant Bluetooth traffic?
- ***RQ4*** Can our attack be used as a vector for exploiting further vulnerabilities?
- ***RQ5*** Can our attack find a significant number of devices that previous attempts miss?
- ***RQ6*** Will the information gathered by our attack be personally identifying to the device owner?

To answer ***RQ1***-***RQ4***, we perform private tests in a controlled environment consisting only of end devices we own. To address ***RQ5*** and ***RQ6***, we must conduct tests against devices we do not control and therefore must take our setup out into the public.

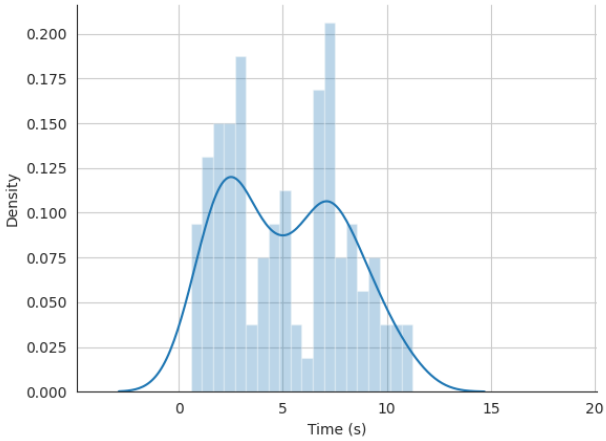

Fig. 6. Probability Density Function (PDF) plot of total attack times. We observe that almost all connections are completed in under 10 seconds from the time we first observe a transmission from the device. Deviations in time come from the variable paging time inherent in Bluetooth Classic and the process of guessing at potential UAPs per device.

C. Private Testing

To prevent our attack from interacting with foreign devices, we performed our private testing within a professional-grade Faraday cage in our lab. This arrangement guaranteed that we did not gather information from devices we did not own. We share an interior shot of our Faraday cage with assorted test devices in Figure 5.

1) Attack Performance: To investigate ***RQ1***, we programmed a timer into our host Ubertooth program. This allowed us to gather timing data for every connection the Ubertooth made while running all of our tests. The timer begins once the transceiver program receives a new LAP from our SDR program and ends after the Ubertooth finishes sending LMP messages to the foreign device, encapsulating the full protocol. A probability density function (PDF) plot representing the results of 100 samples from our timing experiments is shown in Figure 6. We found that our attack achieved an average total time of 5.12 seconds, with connections finishing in as little as 0.57 seconds in the best case. Also, we observe a bimodal shape in this plot with peaks around 2.5 seconds and 7.5 seconds. This occurs because we generally have two addresses to try per device. If our first guess is correct, we can make a connection in under five seconds. However, if our first guess is incorrect, we have to try a second address after hitting a four-second timeout for the first address. This leads to a longer total discovery time that is reflected by the second peak in our graph. Our results demonstrate that our attack is efficient enough to be used against devices transported by pedestrians or vehicles stopped at checkpoints or intersections.

Our discovery time is comparable to that of the traditional inquiry process, although our attack can only attempt connections to a single device at any given moment in our current implementation. This limitation, however, can be mitigated

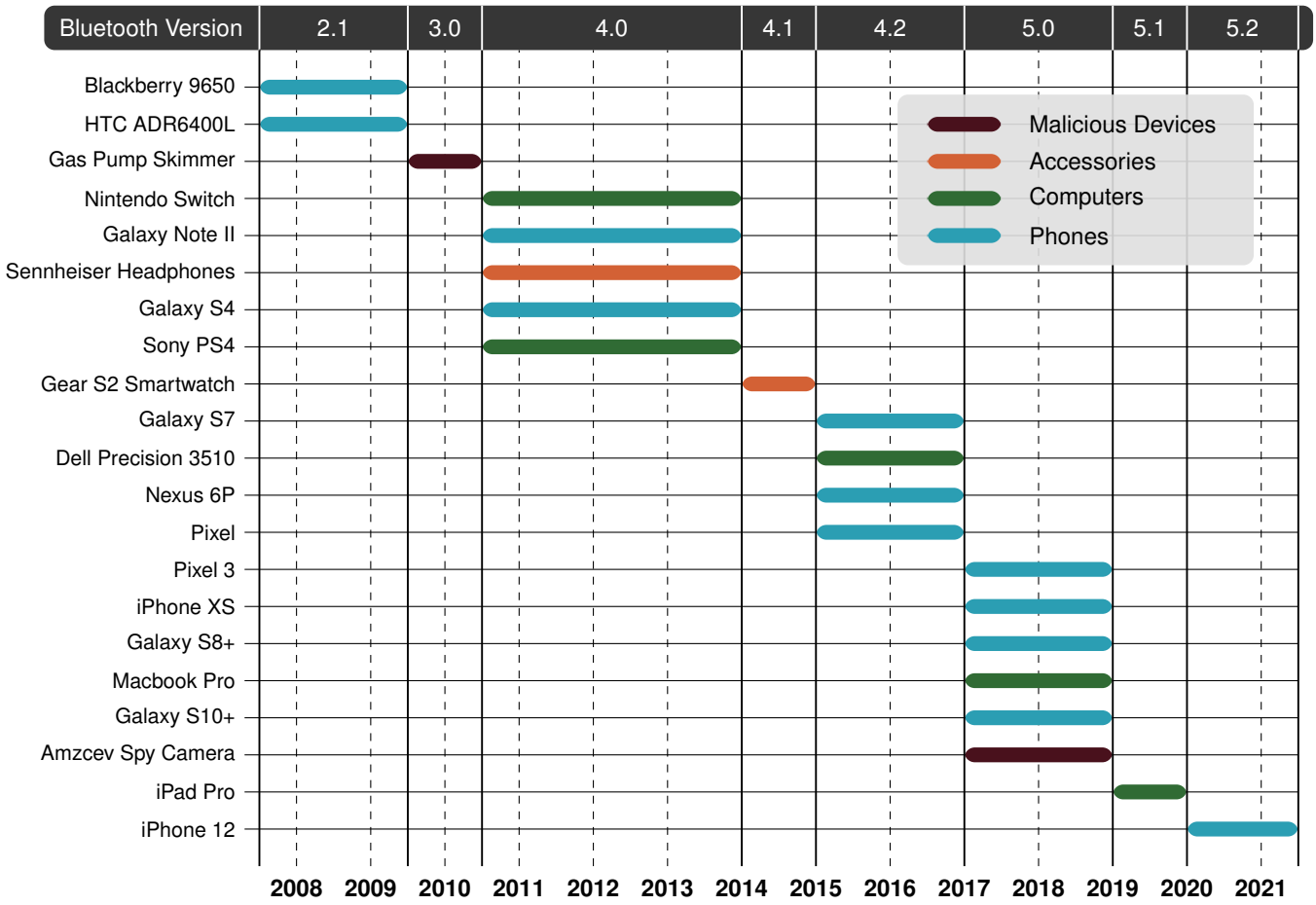

Fig. 7. This timeline incorporates every device we successfully tested our attack against, which we have grouped into phones, computers, accessories, and malicious devices. These devices represent eight different Bluetooth versions, which stretch back to 2007 in release dates. This graphic shows that the *Blue's Clues* attack works against all types of devices, regardless of what version they are using.

by using several transceiver dongles simultaneously. The total time depends on several factors including how quickly the SDR program can narrow down the UAP, how often the target device checks for paging messages, and how many Bluetooth devices are transmitting in the area. Despite targeting multiple devices with different chipsets, we did not observe any correlation between the performance of our attack and the chipset of the target device, likely due to the strict packet structure and timing requirements in the Bluetooth standard.

2) Generality: We employed a collection of various Bluetooth-enabled devices to look into ***RQ2***, a summary of which is shared in Figure 7. In this figure, we show both the variety of victim devices and the Bluetooth versions they run. The oldest version we tested against was Bluetooth 2.1 + EDR, which was accepted by the Bluetooth SIG in 2007, while the newest was Bluetooth 5.2, released in 2020. The lifetime of each Bluetooth version, or the time from inception to replacement, is represented by the colored lines in Figure 7. Our test devices represent 13+ years of Bluetooth versions, *showing that all modern Bluetooth implementations are vulnerable to our attack.* We also report that these devices use Bluetooth chips manufactured by five different companies: Broadcom, Qualcomm, Marvell, Intel, and Bluetrum. While an overwhelming majority of Bluetooth central devices are smartphones, we are interested in what other types of devices can be detected. These other devices included laptops, tablets, game consoles, headphones, and malicious devices such as a spy camera and a gas pump skimmer.[4] This final device class is particularly interesting because we chose to detect devices that are specifically designed *not* to be detected, yet because they are Bluetooth-compliant, they remain vulnerable to our attack just as any Bluetooth device would be. The takeaway from this experiment is that regardless of device type, Bluetooth version, chipset manufacturer, or even device *intent*, any Bluetooth Classic device is vulnerable to our attack.

3) Traffic Experiments: To answer ***RQ3***, we set up eight simultaneous Bluetooth connections between sixteen devices and placed them all into our Faraday cage. We ensured that each device was in Non-Discoverable mode by attempting an inquiry process with a separate Bluetooth device, which found none of them. The test devices included multiple smartphones,

[4]This gas pump skimmer was recovered by law enforcement from a crime scene and shared with our lab.

headphones, speakers, and laptops to simulate a public environment. Our attack discovered all central devices in 41.5 seconds, giving an average of 5.19 seconds of detection time per device. This average is comparable to our overall performance average of 5.12 seconds, proving that our detection time is not significantly affected when the amount of local Bluetooth traffic increases. This result also proves that our attack remains accurate when in the presence of several connections, as it found all possible connections in the isolated environment of the Faraday cage.

We note that previous inquiry-based methods for Bluetooth tracking [21]–[23] would fail to detect *any* devices in this situation. By default, all of the devices we tested against remained in Non-Discoverable mode when not actively searching for new connections. This means that we did not have to change any settings to tailor to our attack, allowing us to conclude that such an environment is representative of a public area.

4) Automatic Vulnerability Exploitation: As our attack automatically detects and establishes connections to foreign devices, we aimed to investigate ***RQ4*** to find out if we could use the established connection for more than just receiving LMP messages. Any vulnerability that does not require knowledge of cryptographic keys could be executed after connection establishment, such as the *BlueFrag* vulnerability (CVE 2020-022) in Android devices [41]. We selected a Google Pixel which had not been updated to patch this vulnerability as our target device. We treated the Pixel as an unknown device by using no other information than could be recovered by our attack. Our attack recovered the name ("Pixel"), the manufacturer ("Qualcomm"), the device type ("Smart Phone"), the Bluetooth version ("4.2"), and the OUI ("Google, Inc."), allowing us to identify the device as a Pixel. Note that while a combination of the manufacturer, device type, Bluetooth version, and OUI are often sufficient in identifying a device (e.g., a smartphone made by Apple is an iPhone), the model name is often found in the device name by default (e.g., "Pixel" or "Steve's iPhone"). In short, the LMP messages reveal the model of the device on the other side of the connection, meaning we can determine which vulnerabilities the device may have without notifying the owner.

For BlueFrag to be successful, the entire Bluetooth address must be known *a priori*, an assumption that is often untrue in attack scenarios. Using the Bluetooth address and device information we recovered with our attack, we were able to identify our target device as a Google Pixel and exploit the BlueFrag vulnerability to trigger a remote denial of service (DOS) attack. To avoid programming the entire BlueFrag protocol into the Ubertooth firmware, we traded out the Ubertooth for a conventional USB Bluetooth dongle[5] which uses the BlueZ Bluetooth stack for Linux [42]. We passed the recovered MAC address into a BlueFrag script [43] which carried out the DOS attack, prompting an error message on the Pixel and forcing the Bluetooth program to restart. Every step of this process could be automated, giving us an automatic vulnerability exploiter that requires no previous knowledge of the victim device.

[5]The Ubertooth could be programmed to carry out the attack itself, but we believe that carrying out the attack on a conventional Bluetooth device was sufficient to support our claim.

D. Public Testing

After gaining approval for a public experiment by the appropriate parties mentioned in our ethics statement, we worked with the faculty within our department to have an official notice of our experiment sent out to an email list containing all personnel with desk space in or keycard access to our building. We chose to conduct the public experiment over two days to (1) maximize the number of devices we could find within our building and (2) provide ourselves with an official second day of experiments in case we faced troubles on the first day. On the first day, our setup ran for three hours on a Friday afternoon. We then ran the setup for six hours the following Monday during peak work hours to try to find as many devices as possible before our allotted approved time expired.

1) Comparison to Previous Methods: To address ***RQ5***, we used a generic Bluetooth chip to perform inquiries for Discoverable devices while our setup was active. In total, we found 23 named Bluetooth devices in our building throughout the two days of testing. With an overwhelming majority of personnel in our building electing to work from home due to COVID concerns, we expected a relatively small number of devices to be transmitting within the building. Out of those 23 devices, our attack was able to find 7 devices that the inquiry process missed, accounting for about a third of the total devices found.

While we cannot make statistically-significant claims based on these results, they do show that *previous inquiry-based methods to location tracking via Bluetooth [21]–[23] miss a major portion of the actual devices in an area*. To produce significant statistics on this metric, the scope of our experiment would have to be widened to include a public space with much greater foot traffic to reach hundreds or thousands of devices. We considered such a study but found that properly informing all participants to align with our ethics statement was not practical in our timeframe given the need to explain the details of the study to every individual participant.

2) Personally-Identifying Information: Finally, we explore ***RQ6*** by manually analyzing the device names we gathered during our public tests. This field often contains the name of the device owner in some way. However, to our knowledge, no study has been conducted to determine how common this occurrence is. From our case study of 23 total devices, we recovered 7 owner first names: 2 from Discoverable devices and 5 from Non-Discoverable devices, accounting for about a third of devices in total. While only 12.5% of Discoverable devices contained a name, 62.5% of Non-Discoverable devices contained a name in our case study. Interestingly, one device appeared to be renamed to the owner's first and last name. This result displays the potential of this attack to *reveal the identity of people in a local area*, something that previous passive approaches [6], [7], [24], [25] cannot offer. We note that while

the device name can be changed arbitrarily, this practice does not seem to be common in our case study. Similar to our result for ***RQ5***, a large-scale test in a public area would yield strong statistics for this question. Once again, we did not aim to contradict our ethics statement for such results.

E. Validation

Throughout the development of our attack, we used the Ellisys Bluetooth Explorer [44] to validate the structure of each of our messages. The Explorer is a solution for over-the-air protocol debugging with 2.4 GHz technologies and can monitor all Bluetooth channels concurrently to capture all Bluetooth traffic. We used this monitor to capture the pairing behavior of several different combinations of Bluetooth devices. This allowed us to check for the traditional ordering and timing of LMP messages, which in turn allowed us to rapidly implement our protocol in our custom Ubertooth firmware to mimic the behavior of standard Bluetooth devices.

VII. Discussion

A. Use Cases

The Blue's Clues attack has both defensive and offensive uses. For instance, the attack could be used to monitor sensitive areas. Locations such as banks and government buildings may wish to monitor and detect all devices, particularly hidden ones. The additional information that our attack recovers would assist in identifying whether the unknown device should be flagged in a manner similar to our suggestion for flagging the spy camera in our evaluation. Similarly, ordinary businesses could deploy our attack to log information throughout the day. Any time theft is committed at the business, the business could immediately reference system logs to look for any devices detected around the time of the crime. If any personally identifiable information was recovered, business owners can report it to the authorities to aid in an investigation.

Conversely, a similar strategy might be adopted in an offensive manner, such as an abusive partner tracking the location of their partner by monitoring their home and/or other locations for a Bluetooth address recovered through our methods. This approach, therefore, allows for a major invasion of privacy. With significant attention being given to abusive tracking via Bluetooth Low Energy tags [45]–[47], our attack demonstrates that Bluetooth Classic is also a potentially dangerous tool for abusers that needs to be addressed. Also, in a malicious setting, the discovery of hidden Bluetooth devices creates a new attack vector. Because our attack recovers a device's manufacturer, type, and Bluetooth version, we were able to demonstrate this by running the BlueFrag attack [41] (which was previously only effective against devices in Discoverable mode). Such an attack is similar to the BlueBorne vulnerability [10], which used WiFi to recover MAC addresses of nearby devices and relied on the Bluetooth MAC address being either identical or similar to the MAC address used in WiFi to establish a Bluetooth connection to the device and execute code. While we only demonstrated an example for one CVE, we believe that any vulnerability that can be executed pre-authentication in Bluetooth can be enhanced to work against Non-Discoverable devices by using our attack.

B. Implications For Standard Writing

Our work contains an important lesson for all standards. Adversaries may eventually gain capabilities beyond anything possible at the time the standard is written, making it incredibly difficult to guarantee security in the long term. This is exemplified by our work exploiting technology that was not available or even considered when the original standard was written two decades ago. One method for combatting this limitation would be an explicit list of assumptions under which the standard provides security.

Along with these explicit assumptions, standard writers could take the evolving nature of technology into account by building flexibility into the standard, thus allowing the standard to evolve in turn. This could include the ability to slot in new cryptographic protocols or allow room for additional authentication protocols. By writing standards with this evolution in mind, writers can potentially observe changes to their initial assumptions before those changes become problematic.

MAC address randomization has been implemented in the newer Bluetooth Low Energy standard [31], as well as in WiFi networks [48]. Standards designers are therefore implementing these lessons learned into newer technologies but can make an additional impact by retrofitting them into existing technologies. We hope that our work enables this approach for Bluetooth Classic.

C. Limitations

Although our attack offers significant improvements over previous solutions, there are still limitations to its capabilities. First, we must be able to capture at least one packet to recover the device address, which means that a device must be actively transmitting for us to detect it. We cannot therefore "ping" for Non-Discoverable devices. This is a limitation shared by all Bluetooth sniffing systems [6], [7] and does not represent a significant issue, as avoiding detection by not using the radio is not a feasible mitigation strategy. A strong adversary can sniff all Bluetooth packets, as mentioned in Section III, to speed up our discovery process by quickly narrowing down address possibilities. However, a connection can be established by brute-forcing all 256 possibilities for the UAP address space after capturing a single packet.

We are also limited to detecting the central device of any connection, as that is the only address used for all active connections. This means that in a piconet with multiple peripheral devices connected to a single central device, all packets will contain the LAP of only the central device. While this does prevent us from detecting all peripherals, the central device is usually the most important to discover. It is often the device most likely to contain personally identifying information, and it is the best target for any vulnerability exploitation. Furthermore, if the central device is compromised, the entire piconet

is compromised. We note here though that if a traditionally peripheral device (e.g., headphones or a speaker) is powered on and attempts to connect to a traditionally central device (e.g., a smartphone or laptop) in range of our setup, we can receive its address. This occurs because any device creating a connection automatically becomes the central device and must perform a role switch with the other device to assume its role as a peripheral device. In the case of headphones connecting to a smartphone, our attack would first capture the address of the headphones, then capture the address of the smartphone after the roles had been switched between the two devices.

While our attack achieved an average discovery time of around five seconds, this time is still not fast enough to discover devices in moving cars or other fast modes of transportation. The attack could not, therefore, be used to gather information on people traveling along highways or in trains without the use of high-gain antennas. Passive approaches can perform quickly enough to handle these situations, but the paging process takes too long on average for our attack to break Non-Discoverable mode under these circumstances. However, our attack can succeed in all other circumstances where devices are not moving at high speeds, including walking or running pedestrians. Note that since our attack contains a passive step, we can perform just as well in fast settings as other work [6], [7], but cannot improve on them without a longer time window.

Finally, we are limited to the Bluetooth Classic protocol, which does not include Bluetooth Low Energy (BLE). BLE, an increasingly popular technology for embedded applications such as asset trackers [1], uses a different addressing scheme. This protocol uses temporary identifiers in lieu of permanent identifiers, which make it incompatible with our attack. However, Bluetooth Classic is still a widespread technology [32], particularly in personal computing devices, meaning our attack is still relevant to the modern wireless ecosystem.

D. Mitigation

1) Proposals: With 4 billion shipments of Bluetooth devices reported in 2020 and a projected 6.4 billion shipments in 2025 [1], patching vulnerabilities at scale is a monumental task. The first step in our attack is demodulating ID packets to recover the LAP as well as error-checking codes that give us information we can use to narrow down the UAP. Preventing this capability would imply removing these IDs from generic Bluetooth packets. However, incorporating IDs into Bluetooth packets is a fundamental design choice of the Bluetooth Classic protocol, so making a major change there would immediately prevent all backward compatibility with existing Bluetooth versions. Purely passive approaches, therefore, will remain a threat against Bluetooth Classic. The newer Bluetooth Low Energy (BLE) standard implements a temporary identifier in place of a static address to solve this problem.

The active stage of our attack can be partially mitigated through two countermeasures: (1) address validation and (2) pre-authentication role-switch prevention. For the first countermeasure, the Bluetooth chip can use the Host Controller Interface (HCI) feature of Bluetooth Classic to pass the MAC address of the device attempting a connection with it to the host, which can keep track of previously paired MAC addresses. Alternatively, the Bluetooth chip itself could employ a small array of saved MAC addresses that it searches whenever a device attempts to connect to it. If the new MAC address is contained in that list, the host can tell the baseband chip to continue with the pairing process. If the MAC address is unrecognized, and the device is not nor has recently been in Discoverable mode, the chip should immediately cease communication with the foreign device. Here we make the observation that the only time a Bluetooth device should expect a pairing procedure from a new device is when it is in (or perhaps, has recently been in) Discoverable mode. This approach is particularly necessary for devices without traditional interfaces (e.g., headphones lack their own screens), without which authentication is generally not possible. While this does not prevent a passive adversary from sniffing a portion of the Bluetooth address, it does prevent an active adversary from querying the device for information, or in the case of devices with no authentication, delivering arbitrary packets (e.g., loud noises to headphones). An adversary could only defeat this mitigation strategy by masquerading as another device that they knew a victim device had previously communicated with. This would require that the adversary observe an FHS packet from a device connecting to the victim device, as well as the LAP and UAP of the victim device. The adversary, therefore, would have to observe either the beginning of a connection or a connection before and after a role switch to recover all of this information, assuming they employ a system that does not have *any* packet loss.

Although this does add some processing overhead to the Bluetooth implementation, it is small enough so as not to affect the protocol. This is similar to the mitigation for the Heartbleed vulnerability [49], which also required the addition of a comparison operation. In that case, the comparison was fast enough that it was undetectable over the network. Our evaluation shows that our mitigation likewise is undetectable by other devices, and only adds a slight overhead on the local device.

Our second proposed countermeasure is rejecting the LMP Role Switch message when it appears before authentication occurs. Rejecting this message would allow the device to retain its NAP and device type information. This prevents our attack from gathering the full Bluetooth MAC address (a permanent and unique identifier), the host manufacturer (e.g., Apple or Google), and the device type (e.g., smartphone). Lacking this information limits our attack as a means of location tracking, and prevents us from quickly determining the device model, which we use in our automatic vulnerability exploit experiment. Recent work has taken advantage of overly-permissive pre-authentication role switching in Bluetooth Classic for other purposes [26], [27], further proving the need for this change.

Finally, users may wish to limit the use of personally identifiable information such as device names. Many devices

including iPhones name devices after the user (e.g., *Steve's iPhone*) by default. Our IRB-approved test confirmed that this practice was common and demonstrated that a surprising 78% of phones in the building had the user's first name as part of the device name. We note that while our study is small due to COVID conditions, we believe that it is a reasonable approximation for common user behavior in the larger public. That said, we recognize that it is not entirely practical to remove use names from device identifiers, as many users want to be able to easily identify their phones.

2) Evaluation: We identified address validation as our most effective countermeasure because it prevents *any* information from being leaked to an unknown device and therefore chose to evaluate this option. To verify that our address validation proposal is valid, we implemented the necessary fix to our open-source Ubertooth firmware [39], which is written in *C*. The Uberooth uses an LPC175x ARM Cortex-M3 microcontroller [50] running at 50 MHz. We first created a standard array of size 262[6] to act as our small database of saved MAC addresses. We added an entry to the array representing the MAC address of one of our test devices (iPhone 12) and encapsulated the function call to establish a connection with a conditional statement. This statement only allows that function to be called under two conditions: (1) if the MAC address of the device attempting to establish a connection with the Ubertooth matches an entry in our small database or (2) if the device is currently Discoverable. If neither of those conditions is true, communication simply ceases between the two devices. For reference, we provide pseudocode for this process in Algorithm 1 which exhibits all three possible conditions.

Algorithm 1 Address Validation Countermeasure

```
if MAC ∈ Database then
    Accept LMP Transaction
else if MAC ∉ Database and Discoverable then
    Accept LMP Transaction
else
    Reject LMP Transaction
end if
```

We used a separate Bluetooth device (Huawei Nexus 6P) to act as our attacker and paged the Ubertooth from that device. The Ubertooth saw that the MAC address of the Nexus was not in its database and refused the connection. When we performed the same paging procedure with the iPhone, the Ubertooth matched the iPhone's MAC address to the entry that we made in our database and allowed the connection to continue.

We evaluated the total time taken by our countermeasure by determining the amount of elapsed clock cycles from the beginning of searching the database to the end of the conditional statement. We inserted the correct MAC address at the very end of the array to ensure that we were evaluating a worst-case scenario. In all 50 of our tests, we recorded exactly 2659 clock cycles to perform the MAC address check on the Ubertooth, which equates to 50.18 μs. To ensure this value did not slow down the protocol, we recorded the time taken between a device receiving an FHS packet (which includes the MAC address) and its acknowledgment ID packet using our Ellisys Bluetooth Explorer [44]. For reference, this process is outlined in Figure 3 as the final two steps in *Paging Procedure*. Over eight runs, each with different devices, we observed an average of 625.28 μs with an extremely tight standard deviation of 0.63 μs. This means that our fix can be evaluated significantly faster than is necessary to keep up with the Bluetooth protocol. Performing the search before sending the acknowledgment packet also prevents the attacker from confirming the device UAP on top of preventing leakage of LMP information and the full MAC address.

If the search cannot be performed before the acknowledgment must be sent, a device could simply perform it before it must respond to the first LMP request, which will be the first message it receives after the paging procedure ends. This would not guard the UAP, but would still protect against leakage of LMP information and the full MAC address. We observed an average time between receiving the FHS packet and the first LMP request of 15.31 ms over eight runs with different devices. These tests produced a large standard deviation of 22.16 ms, which suggests that the time between the end of the paging procedure and the beginning of LMP transactions is not strictly set by the Bluetooth standard. Nevertheless, our evaluation time of 50.18 μs is significantly faster than even the smallest time difference we observed from these tests (3.75 ms). Finally, we note that observing an exact number of clock cycles over repeated tests for an embedded device such as the Ubertooth is not surprising because the Ubertooth is only performing a single task during that period.

E. Disclosure

Around the time of submission, we disclosed the vulnerability to the Bluetooth SIG. They have confirmed that our method is a means of identifying the full Bluetooth Address of a BR/EDR radio that is Connectable but Non-Discoverable. As such, they are interested in providing mitigation for existing devices as our process is faster than any previously known approach for discovering Non-Discoverable devices. They discussed a mitigation strategy that has a device set its name to an empty string when in Non-Discoverable mode to limit data leakage. This, however, would not prevent the recovery of the Bluetooth address, nor the device's type, manufacturer, version, etc. Our mitigation technique, as outlined in Section VII-D, would prevent the disclosure of all this information without impacting the functioning of the Bluetooth protocol. We have reserved CVE number **CVE-2022-24695** for our disclosure. The representative from the SIG has indicated that they intend to continue to investigate the issue internally; however, as we are not a member company, we will not be able to see any additional potential mitigations until they are publicly released.

[6]Bluetooth Classic only allows for up to 262 devices in a piconet (7 active and 255 on standby) [31], so we believe that a small database size of 262 is reasonable.

VIII. Related Work

Bluetooth has undergone thorough security analyses over the past 20 years, ranging from cryptanalysis to practical over-the-air attacks. Early work targeted vulnerabilities in the cryptography of the standard, specifically the E_0 stream cipher and Bluetooth PIN [9], [11]–[17]. Location leaks in Bluetooth's Discoverable mode were first discussed by Jakobsson et al. in the early 2000s [9]. In this work, the authors revealed that a specific Bluetooth device, and therefore the device owner, could be tracked using a network of Bluetooth devices that perform the standard inquiry process to reveal nearby connectable Bluetooth devices in Discoverable mode. This idea was evaluated in different settings spanning from shopping malls [22] to entire cities [21], [23]. Both a distributed network of Bluetooth sensors [21], [22] and a single sensor mounted on a moving vehicle [23] were used in these studies, which all found success in tracking individuals. Such an approach is thrawted by Bluetooth's Non-Discoverable mode, as it relies on devices that respond to inquiry requests.

Jakobsson et. al. also considered passive attacks against Bluetooth's Non-Discoverable mode by suggesting an attack in which an adversary intercepts an ID packet containing a Bluetooth Channel Access Code (CAC) used after the establishment of a piconet. However, this attack was not demonstrated to be practical until the *BlueSniff* work in 2007 [7]. From Jakobsson's attack, Wong et al. [18] suggested that Bluetooth adapt pseudonyms similar to GSM TMSIs to prevent leakage of permanent Bluetooth addresses to passive adversaries. They also mention that the location privacy of a peripheral device can be compromised by an external device that captures a paging message addressing that device. A few years later, the *BlueSniff* work by Spill et al. [7] provided the first open-source Bluetooth sniffer code *gr-bluetooth* in GNU Radio. They used this technology to eavesdrop on several neighboring Bluetooth channels, developing a technique for *de-whitening* Bluetooth packets to recover the LAP and reversing HEC and CRC generation algorithms to recover the UAP. These achievements paved the road for practical Bluetooth security research, while also revealing weaknesses in location privacy with the Non-Discoverable mode while in the presence of a sniffer device. Furthermore, they suggest a method of determining the NAP of a device by consulting the OUI list [33] of MAC addresses, which can quickly narrow down the list of potential NAPs. This was the first practical proposal for recovering the entire Bluetooth MAC address. That same year, efforts were underway to recover the Bluetooth address of devices that were not transmitting through a brute force attack. While the naive approach would theoretically take 1.4 years of continuous transmission on a single radio to recover the LAP, such an attack could be reduced to 7 days with 79 radios, one for each Bluetooth channel [8]. This effort was further refined by Cross et el., [5], who used a more efficient method to brute force the address in less than 24 hours, given 79 radios. While this approach is theoretically sound, it does assume an exaggerated threat model. Moreover, recent approaches to recovering the Bluetooth MAC address are significantly more performant.

Our work benefits primarily from the wideband Bluetooth skimming performed by Cominelli et al. [6]. In that work, the authors use the information in a de-whitened header to narrow the possible UAPs down from 255 to 2. The LAP and small set of UAPs worked as an identifier that was stable and could be tracked over time. Although it is statistically likely that this identifier is unique, two devices could have the same UAP and LAP, but differ in NAP, preventing [6] from definitively tracking a single device. We build on their methods with an active component that affords us the capabilities of all previous attacks on both Discoverable and Non-Discoverable devices. Additionally, we can consider our approach as a door for exploiting vulnerabilities or injecting packets into Bluetooth devices which lack mutual authentication (e.g., playing music on a speaker).

IX. Conclusion

In this work, we present Blue's Clues, the first complete break of the Non-Discoverable mode of Bluetooth Classic. Our attack uses a two-step process for device discovery by first recovering a portion of the Bluetooth MAC address, then using the address fragment to establish a connection to the corresponding device. During the connection, we can query any Non-Discoverable device for information that is normally shared freely by Discoverable devices. Our attack shows how this information can be used to track individuals or expose a device to exploits that traditionally target systems in Discoverable mode. We implemented our attack on consumer hardware, using open-source host code and device firmware to carry out all processes. By testing the attack on a wide variety of devices, we showed that all Bluetooth Classic standard-compliant devices are vulnerable.

Additionally, we demonstrated the impact of our attack through experiments both in private and public settings, exhibiting the benefits of our attack compared to past work in Bluetooth location tracking. Namely, Blue's Clues both finds devices that inquiry-based methods miss and recovers personally-identifying information on device owners that passive approaches cannot produce. Finally, we propose and evaluate mitigation methods for defending against our attack by rejecting connections from unknown addresses when the device has not recently been in Discoverable mode, rejecting pre-authentication role switches, and removing personally-identifying information from device names.

Acknowledgments

We would like to express our gratitude to our sponsors for enabling this research. This work was supported by the US National Science Foundation grant CNS-1815883, the Office of Naval Research grant ONR-OTA N00014-20-1-2205, and the Air Force Research Laboratory award AFRL FA8650-19-1-1741.

REFERENCES

[1] "Bluetooth Market Update," https://www.bluetooth.com/bluetooth-resources/2021-bmu/, 2021.

[2] "Bluetooth Core Specification 1.0B," Bluetooth SIG, Tech. Rep., 1999.

[3] J. Padgette, J. Bahr, M. Batra, M. Holtmann, R. Smithbey, L. Chen, and K. Scarfone, "Guide to Bluetooth Security," NIST, Tech. Rep. 800-121 Rev 2, 2017.

[4] "Security Tip (ST05-015) Understanding Bluetooth Technology," https://www.cisa.gov/tips/st05-015, 2021.

[5] D. Cross, J. Hoeckle, M. Lavine, J. Rubin, and K. Snow, "Detecting Non-Discoverable Bluetooth Devices," in *Proceedings of the International Conference on Critical Infrastructure Protection*, 2007.

[6] M. Cominelli, F. Gringoli, M. Lind, P. Patras, and G. Noubir, "Even Black Cats Cannot Stay Hidden in the Dark: Full-band De-Anonymization of Bluetooth Classic Devices," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2020.

[7] D. Spill and A. Bittau, "BlueSniff: Eve Meets Alice and Bluetooth," in *Proceedings of the USENIX Conference on Offensive Technologies (WOOT)*, 2007.

[8] K. M. J. Haataja, "Three Practical Bluetooth Security Attacks Using New Efficient Implementations of Security Analysis Tools," in *Proceedings of the IASTED International Conference on Communication, Network and Information Security*, 2007.

[9] M. Jakobsson and S. Wetzel, "Security Weaknesses in Bluetooth," in *Proceedings of the Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, 2001.

[10] B. Seri and G. Vishnepolsky, "BlueBorne," Armis, Tech. Rep., 2017.

[11] M. Hermelin and K. Nyberg, "Correlation Properties of the Bluetooth Combiner Generator," in *Proceedings of the International Conference on Information Security and Cryptology (ICISC)*, 1999.

[12] Y. Lu and S. Vaudenay, "Cryptanalysis of Bluetooth Keystream Generator Two-Level E0," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2004.

[13] ——, "Faster Correlation Attack on Bluetooth Keystream Generator E0," in *Proceedings of the International Cryptology Conference (CRYPTO)*, 2004.

[14] Y. Lu, W. Meier, and S. Vaudenay, "The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption," in *Proceedings of the International Cryptology Conference (CRYPTO)*, 2005.

[15] Y. Shaked and A. Wool, "Cracking the Bluetooth PIN," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2005.

[16] F.-L. Wong, F. Stajano, and J. Clulow, "Repairing the Bluetooth Pairing Protocol," in *Security Protocols Workshop*, 2005.

[17] Y. Shaked and A. Wool, "Cryptanalysis of the Bluetooth E0 Cipher Using OBDD's," in *Proceedings of the International Conference on Information Security (ISC)*, 2006.

[18] F.-L. Wong and F. Stajano, "Location Privacy in Bluetooth," in *Proceedings of the European Workshop on Security and Privacy in Ad-hoc and Sensor Networks (ESAS)*, 2005.

[19] J. Dunning, "Taming the Blue Beast: A Survey of Bluetooth Based Threats," *IEEE Security & Privacy*, vol. 8, no. 2, pp. 20–27, 2010.

[20] S. S. Hassan, S. D. Bibon, M. S. Hossain, and M. Atiquzzaman, "Security Threats in Bluetooth Technology," in *Proceedings of the IFIP TC-11 International Information Security and Privacy Conference*, 2018.

[21] M. Versichele, L. de Groote, M. Claeys Bouuaert, T. Neutens, I. Moerman, and N. Van de Weghe, "Pattern Mining in Tourist Attraction Visits Through Association Rule Learning on Bluetooth Tracking Data: A Case Study of Ghent, Belgium," *Tourism Management*, vol. 44, pp. 67–81, 2014.

[22] D. Oosterlinck, D. F. Benoit, P. Baecke, and N. V. de Weghe, "Bluetooth Tracking of Humans in an Indoor Environment: An Application to Shopping Mall Visits," *Applied Geography*, vol. 78, pp. 55–65, 2017.

[23] M. Chernyshev, C. Valli, and M. Johnstone, "Revisiting Urban War Nibbling: Mobile Passive Discovery of Classic Bluetooth Devices Using Ubertooth One," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1625–1636, 2017.

[24] B. Rodrigues, C. Halter, M. Franco, E. J. Scheid, C. Killer, and B. Stiller, "BluePIL: A Bluetooth-Based PassIve Localization Method," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021.

[25] Y. Wang, X. Yang, Y. Zhao, Y. Liu, and L. Cuthbert, "Bluetooth Positioning Using RSSI and Triangulation Methods," in *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, 2013.

[26] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen, "The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR," in *Proceedings of the USENIX Security Symposium*, 2019.

[27] D. Antonioli and N. O. Tippenhauer, "BIAS: Bluetooth Impersonation AttackS," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2020.

[28] T. Claburn, "Oh Cool, the Bluetooth 5.1 Specification is Out. Nice. *Control-F* Master-Slave... 2,000 Results," https://www.theregister.com/2019/02/01/bluetooth_updates_tech/, 2019.

[29] E. Landau, "Tech Confronts Its Use of the Labels 'Master' and 'Slave'," https://www.wired.com/story/tech-confronts-use-labels-master-slave/, 2020.

[30] C. McKenzie, "Master-Slave Terminology Alternatives You Can Use Right Now," https://www.theserverside.com/opinion/Master-slave-terminology-alternatives-you-can-use-right-now, 2019.

[31] "Bluetooth Core Specification 5.2," Bluetooth SIG, Tech. Rep., 2019.

[32] S. Karr, "Consumer Survey Highlights Necessity of Bluetooth Technology," https://www.bluetooth.com/blog/consumer-survey-highlights-necessity-of-bluetooth-technology/, 2016.

[33] "IEEE SA - Registration Authority," https://standards.ieee.org/products-programs/regauth/, 2020.

[34] "Btsniffer," https://github.com/bsnet/btsniffer, 2020.

[35] M. Huges, "What is Bluetooth 5? Learn about the Bit Paths Behind the New BLE Standard," https://www.allaboutcircuits.com/technical-articles/long-distance-bluetooth-low-energy-bit-data-paths/, 2017.

[36] "USRP B210 USB Software Defined Radio (SDR)," https://www.ettus.com/all-products/ub210-kit/, 2020.

[37] M. Ossmann, "Project Ubertooth: Discovering Bluetooth Devices," http://ubertooth.blogspot.com/2012/10/discovering-bluetooth-devices.html, 2012.

[38] ——, "Ubertooth One," https://greatscottgadgets.com/ubertoothone/, 2014.

[39] ——, "Ubertooth," https://github.com/greatscottgadgets/ubertooth, 2020.

[40] "USRP Hardware Driver and USRP Manual: Building and Installing UHD from source," https://files.ettus.com/manual/page_build_guide.html, 2020.

[41] J. Ruge, "BlueFrag," https://insinuator.net/2020/04/cve-2020-0022-an-android-8-0-9-0-bluetooth-zero-click-rce-bluefrag/.

[42] "BlueZ," http://www.bluez.org/, 2000.

[43] "CVE-2020-0022," https://github.com/Polo35/CVE-2020-0022, 2021.

[44] "Ellisys Bluetooth Explorer All-in-One Bluetooth® Analysis System," https://www.ellisys.com/products/bex400/, 2022.

[45] M. Chin, "AirTags are Dangerous — Here's How Apple Could Fix Them," https://www.theverge.com/2022/3/1/22947917/airtags-privacy-security-stalking-solutions, 2022.

[46] A. Li, "Google Looks to be Building Bluetooth Tracker Detection Directly into Android," https://9to5google.com/2022/03/29/android-bluetooth-tracker-detection/, 2022.

[47] K. Hill, "I Used Apple AirTags, Tiles and a GPS Tracker to Watch My Husband's Every Move," https://www.nytimes.com/2022/02/11/technology/airtags-gps-surveillance.html, 2022.

[48] M. Burton, "What is Wi-Fi MAC Randomization and How Does it Handle Privacy?" https://www.extremenetworks.com/extreme-networks-blog/wi-fi-mac-randomization-privacy-and-collateral-damage/, 2020.

[49] "Heartbleed Bug," https://heartbleed.com/, 2020.

[50] "LPC1700 Series," http://www.nxp.com/pages/:MC_1403790745385.