A Bluetooth Keyboard Attack

Torrey Cuthbert, Andrew Gontarek, Elizabeth Jensen, and Peter Robbins

{cuthb016, gont0005, jense924, robbi094}@umn.edu

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455

***Abstract*—Bluetooth devices have become increasingly popular and are expected to surpass two billion devices sold by 2010. Bluetooth provides communication between devices, such as cell phones and headsets, or computers and robots, among other things. Bluetooth keyboards and mice are also very popular, making people more mobile when working at their computers. Unfortunately, the Bluetooth protocol also has some significant security holes. We present here our research into an implementation-level man-in-the-middle attack on a Bluetooth keyboard, which allows us access to all keystrokes typed on the keyboard, and thus provide us with passwords and other useful information about a user. We also discuss ways to mitigate or prevent such an attack.**

I. Introduction

Bluetooth is an open wireless protocol for exchanging data over short distances (using short length radio waves) from fixed and mobile devices, creating personal area networks (PANs). Mobile phones, headsets, laptops, PCs, PDAs, printers, digital cameras, mice and keyboards are all common devices that use Bluetooth technology. The communication or data exchanged between these devices can be short, such as a command to turn on the motors for a robot; or, when used with a hands-free handset, for example, as long as the length of an entire phone conversation. The popularity of Bluetooth devices are on the rise. By 2006, one billion Bluetooth capable devices had been shipped, and it is anticipated that there will be two billion Bluetooth devices sold by the end of 2010 [1]. Due to their popularity and wide range of use, it is important to keep Bluetooth communication channels secure [2].

When the first devices were released in 2000 [1], there were still significant vulnerabilities in the technology [1-3]. These security issues will be discussed further in Section II where we examine related works and previous exploits. For the most part, these previous vulnerabilities were present in the protocol and are generally addressed in later Bluetooth specifications. In this work, we will show that Bluetooth communication is still vulnerable to attacks due to vulnerabilities based on how the protocol is implemented in an operating system. Here we focus on an attack on Bluetooth keyboards, though elements of the attack can be implemented elsewhere. We chose to show vulnerabilities in keyboards, since sensitive and private information is usually inputted this way. Our contribution is an implementation-level man-in-the-middle attack on a connection between a Bluetooth keyboard and a computer, which provides the same functionality as a keylogger on a Bluetooth keyboard, and which will show up as nothing more than a device malfunction. We also discuss ways in which a user can protect themselves against such an attack.

The remainder of this paper is organized as follows. Section II details the Bluetooth protocol and related works on Bluetooth security vulnerabilities and exploits. Section III describes

our attack. Experiments and their results are presented in Section IV. Recommendations and countermeasures for protecting against similar attacks are provided in Section V. Our conclusions are documented in Section VI.

II. Related Works

A. Bluetooth Protocol

Bluetooth is a wireless protocol designed for use in the low-power, low-speed implementation that is often necessary in peripherals such as headsets, keyboards, and mice. The Bluetooth protocol stack can be divided into four layers--Bluetooth Core Protocols, Cable Replacement Protocols, Telephony Control Protocols, and Adopted Protocols. The core protocol layers, which include the Bluetooth radio, are entirely Bluetooth specific and are required by most Bluetooth devices. The remaining three are only used as needed and for the purposes of this research are excluded from discussion.

Everything in Bluetooth is run over the Radio Layer, which defines the requirements for a transceiver operating in the 2.4GHz band. The radio layer also defines sensitivity levels of the transceiver, establishes the requirements used in "Spread-spectrum Frequency Hopping" [4], and classifies Bluetooth devices in three power classes by device operation range (Class 1, 100m; Class 2, 10m; and Class 3, 10cm). Bluetooth utilizes a per-packet 79 channel frequency hopping scheme. Although designed as a mechanism to ease interference on the Bluetooth frequency spread, the channel hopping makes eavesdropping on Bluetooth connections difficult to execute in practice, since current Bluetooth packet sniffing suites are currently prohibitively expensive.

Above the Radio Layer sits the Baseband Layer, which is Bluetooth's physical layer. It is used for tasks such as creating link connections with other devices. It controls device addressing, how devices find each other (channel control) through paging and inquiry methods, power-saving operations, flow control and synchronization. The next layer, the Link Manager Protocol (LMP), controls link setup, authentication, link configuration and other protocols. It finds other link managers within the area and communicates with them via its protocol. Finally we have the Logical Link Control and Adaptation Protocol (L2CAP) and the Service Discovery Protocols (SDP). The L2CAP provides connection-oriented and connectionless data to the upper layer protocols by plugging into the physical layer (baseband). It transports packets up to 64 kilobytes in size. The SDP simply enables applications to detect available services and learn about them, and relies on the L2CAP to do so.

In addition to the Bluetooth core and other layers, the Bluetooth specification defines a Host Controller Interface (HCI) which provides a command interface to the Baseband Layer and LMP for control and to receive status information. It consists of three parts: firmware, driver, and host controller transport layer. Although the HCI is typically positioned below the L2CAP in the stack, this position is not mandatory and can be implemented above the L2CAP.

A Bluetooth connection is initiated by a "master" device, making a connection request to the "slave" device. If using encryption, they then proceed through a "pairing" process that uses Diffie-Hellman key exchange along with a PIN to establish a secure link key for the connection. This link key is stored by both devices with the MAC address of the other device. Once paired, all traffic between the master and the slave is encrypted. This pairing process is the most often targeted section of Bluetooth devices.

B. Bluetooth Security Vulnerabilities

Bluetooth devices were first released in 2000 [1]. Some of the first research into security holes in the technology was published the following year, in 2001. Three attacks were detailed in [5]. The attacker could do an offline, exhaustive search of PINs, or execute a man-in-the-middle attack and then have the ability to eavesdrop or interfere with a connection. The second attack was able to give the attacker the geographic location of the victim's device, which could lead to blackmail or some form of espionage. The third attack was much lower level, and revolved around breaking the cipher. All three of these attacks were made possible by holes in the protocol itself. The authors suggested fixes to the

protocol, such as changing the cipher to AES, which would mitigate the threat from these particular threats [5].

The weaknesses in the PIN were still present in 2005, when another method of cracking the code was presented [3] for Bluetooth 2.0, which had been released the previous year. Here, the authors were able to crack the PIN of 4 digits in 0.063 seconds on a Pentium IV 3Ghz HT. In the same year, [6] listed 24 different attacks, including two different PIN attacks. These attacks range from brute-force and denial-of-service attacks (DoS attacks) to replay and man-in-the-middle attacks. There are even attacks on entire Bluetooth networks, rather than just a single pairing. Of the 24 attacks, however, very few exploit a hole that is not a part of the protocol. The only one that was obviously not related to the Bluetooth protocol was the one in which keys were disclosed through using false Bluetooth dongles, USB drives, or malicious software. In [7], Wong, Stajano and Clulow proposed a way to fix the pairing vulnerability in the Bluetooth protocol. They first showed that they could recover the link key shared by two devices, and with it decrypt and encrypt data that passed over the connection. They then presented a pairing protocol with better security using asymmetric cryptography with minimal effect on usability of even handheld devices. The DoS attacks are more closely studied in [8], and the main vector of DoS attacks is to overload the Bluetooth device as a whole.

There were some more formal analyses of the Bluetooth protocol in 2007, including [9]. In this work, the authors performed an analysis using a cryptographic protocol verifier to test the Bluetooth protocol for holes in security. They found the usual one involving the PIN, and also found holes in a newly suggested pairing mechanism. They went on to show how that pairing protocol could be fixed to have the security assurances that were desired. 2007 also saw the first open source Bluetooth sniffer, [10] which used a USRP to eavesdrop on the Bluetooth communication (with some limitations due to the hardware capabilities). The authors concluded, from their research, that it would not be possible to eavesdrop on Bluetooth connections without the expensive equipment they used, so it was unlikely that this would be a normal vector of attack. However, Max Moser found that it was possible to use a commercially available dongle (at the cost of $30) to eavesdrop on Bluetooth communications and that the majority of specialized Bluetooth sniffing packages rely on the software rather than the hardware to sniff Bluetooth packets [11]. From this, it is a logical conclusion that an adversary could build a Bluetooth packet sniffer, since expensive hardware is not the limitation.

Unfortunately, Bluetooth security took a step backward, according to Lindell in his 2008 paper [12]. He exposed two new attacks that had not been possible in version 2.0, but which were introduced in version 2.1. The most startling was the one in which the pairing protocol, which has been at the root of many previous security holes, would fail entirely, and leak the PIN. Additional vulnerabilities in Bluetooth 2.1 are pointed out in [13]. Both [2] and [14] detail further attacks that are still possible in the current versions of Bluetooth (2.0 and 2.1).

Nearly all of these attacks have made use of holes in the Bluetooth protocol itself. We have found only two papers in which key logging was one of those attacks [2, 15]. Our approach to the problem is thus different from the majority of the previous body of work on three counts. First, we chose to implement a man-in-the-middle attack that exploits the implementation of Bluetooth and how it is maintained by the user, rather than exploiting a hole in the protocol, as has been done previously. Second, we wish to implement our attack on the connection between a computer and a Bluetooth keyboard, something which has been mostly overlooked, as connections to cell phones are a more popular device. Third, we use a denial-of-service attack that targets only the connection between the computer and the keyboard, leaving all other Bluetooth devices connected to the computer functional.

III. Implementation

The first step of our attack is to find the Bluetooth MAC address of the targeted keyboard and computer. This can be done with relative ease

through one of two methods. In the first, the attacker could run a brute-force attack using a multi-threaded version of @stake's RedFang to extract the MAC addresses in approximately 90 minutes [16]. In the second, the attacker could use a Bluetooth packet sniffer (which is shown to be possible with a regular Bluetooth dongle [11]) to look at the Bluetooth packet headers and extract the source and destination addresses. Once the addresses are found, the attack can begin.

We found that some Bluetooth stacks will throw away a link key if it receives a connection request from a MAC address for which it has pairing information. When the Bluetooth stack sees this, the link key for that address is thrown out and a new one is forged through the pairing process. This happens transparently, since the device will store the PIN used to create the link key. On some of the operating systems we tested against, it did not matter if the victim's computer had an already-existing connection with a device at that MAC address; if the connection request came from a paired MAC address (even if it was not on the existing link channel) the link key would be thrown out and a new one created on the existing link channel. So, by simply setting the MAC address of our computer's device to that of the victim's keyboard and making a connection request to the victim's computer, we were able to force the victim's computer and keyboard to re-pair. By doing this repeatedly, the victim's computer is constantly throwing away Bluetooth link keys and re-pairing with the keyboard so that no traffic besides the re-pairing communication can get through. This DoS attack renders the keyboard non-functional without interfering with other Bluetooth traffic, so that the average user will assume that it is a keyboard malfunction rather than a computer malfunction.

We assume that, when confronted with a non-functional device, the average user will follow whatever steps made the device function previously in an attempt to make the device function one again. In the case of our attack, the user will press the pairing button on the keyboard and attempt to start the pairing process over. This puts the keyboard into discoverable and pairable mode. When the keyboard is observed to be in this mode, the attacker can then pair in unencrypted mode, which requires no pin entry on the keyboard. After this, whatever is typed on the victim's keyboard is sent to the attacker. The attacker can then broadcast that their computer is a Bluetooth keyboard with the same name, MAC address, and capabilities of the victim's keyboard. Since the victim's keyboard is no longer in discoverable mode, the victim will only see the attacker's device, which has the appearance of their own keyboard. The victim will then try to connect with the attacker's “keyboard” and type the correct PIN into the keyboard. Since the attacker is connected to the keyboard already, they simply use whatever the victim typed on the keyboard to complete the pairing process. After this, the attacker now has a connection with both the victim's keyboard and computer and can then read and forward key-presses from the keyboard to the victim's computer, serving as a transparent keylogger.

IV. Experimentation

A. Environment

In our experiment we did not deal with discovering the MAC addresses of our target Bluetooth devices since previous work has shown that this is possible through packet sniffing or brute-forcing the address space. Instead, since we were in possession of the MAC addresses of both the target keyboard and computer we proceeded with our attack, assuming that an attacker would be able to acquire the MAC addresses using previous methods.

Our attacker system was a notebook running Linux with the BlueZ Bluetooth stack and two Bluetooth cards. We changed the MAC address of one of the cards to the address of the target keyboard and used standard BlueZ utilities to launch a DoS attack on the victim's keyboard link. Using a script invoking "hcitool" we made repeated connection requests to the victim's computer at one second intervals. When the pairing button on the target keyboard was pressed, we ended the DoS attack and paired with the keyboard with our second Bluetooth card using an unencrypted connection. We set up the first card

to then look like the keyboard and placed it into discoverable and pairable mode. To the victim's computer, our computer had every appearance of being the victim's Bluetooth keyboard. When the victim sent a pairing request to our "keyboard" it was handled by the standard OS pairing request handler as a PIN entry dialog box. When the PIN was entered on the keyboard (which was connected to the attacker's computer as a standard keyboard) it was entered into the dialog prompt and the pairing between our "keyboard" and the victim's computer was successful. From there we only needed to read the input from the keyboard and send our own Human Interface Device commands through the connection to the victim. It should be noted that, apart from changing the device Bluetooth address (which is a fairly simple bit of BlueZ library programming), the entire attack was carried out using standard BlueZ command line utilities.

B. Results

The results of our attack varied depending on the target operating system. The DoS was ineffective against Windows 7 and Windows XP SP2 target systems, but functioned perfectly against Linux BlueZ and Windows Vista target systems. On both Linux and Vista, the DoS created continuous reconnection between the keyboard and computer; no user input from the keyboard made it through. Additionally, the DoS did not interfere with the victim's communication with other devices. For example: we noticed no irregular behavior while using a Bluetooth mouse on the target computer during the DoS attack. For all targeted operating systems, we had no problems with the attack once the pairing button on the keyboard was pressed. We could easily pair with the keyboard and then appear to be a keyboard to the targeted system and handle the pairing request. We were unable to complete an application to actually forward keystrokes from the keyboard to the victim's computer due to some plug-and-play driver peculiarities, but since we were able to get a connection with the keyboard, a valid pairing with the victim's computer, and a partially functioning keystroke-forwarding application, we conclude that this element, while not simple, is not a prohibitively difficult bit of software for an attacker to build.

C. Discussion

The most important and dangerous part of our attack is the ability to make a Bluetooth device appear to be malfunctioning by sending only one command per second. The Bluetooth specification section 4.2.1.2, *Claimant has no link key* [17], allows a Bluetooth device to forget a link key, so the victim seeing a connection request stating that the device does not have a link key is *required* to be seen as legitimate and a new link key created. How this is implemented is up to the operating system's Bluetooth stack. It appears that both Windows 7 and XP will ignore a packet on a new connection from an address that has an existing connection, but BlueZ and Windows Vista handle the packet as if it came from the valid connection. While we used this weakness to launch a DoS attack against a keyboard, this could be a point of entry for other packet injection attacks.

The Bluetooth 2.1 specification section 5.1, *Repeated Attempts* [17], attempts to address DoS attacks by requiring an exponential delay between successive authentication failures. Since our attack results in successful reconnection, there is no delay and every action is valid within the Bluetooth specification. The Bluetooth 3.0 specification makes mention of repeated successful pairings, requiring the device to change its public key after 10 successful pairings. This, however, seems designed to address extracting information regarding the PIN and link key as in [12], and not the DoS weakness we have explained.

Performing the actual man-in-the-middle attack was relatively simple at the implementation level we have described. It is well-known that a computer can masquerade as a keyboard and that a computer can pair with a keyboard in pairable mode. While this is a known danger, we remove the necessity for an attacker to be present to intercept the initial pairing of target devices by providing a method (on vulnerable systems) to make it appear that only a single device is malfunctioning. If the user, confronted with a

malfunctioning device, goes through the same steps that made the device functional before (the initial pairing process), the attacker has forced the user to create the exact conditions that they can then exploit.

V. Recommendations

Eliminating the denial-of-service vulnerability reduces the effectiveness of our attack by forcing the attacker to be present at the initial pairing, which is highly unlikely. Since the weakness is not in the Bluetooth protocol and rather in how it is implemented in an operating system, a fix could be easily made available to users as a security patch or an updated version of software. The solution to fix vulnerable Bluetooth implementations is simple: either allow only one connection per MAC address or enumerate connections from the same MAC address. By disallowing a new connection from a connected MAC address, the attacker's connection request would be ignored and everything would function as normal. If the connections were enumerated, the attacker's connection attempt would fail and begin an exponential delay between attempts as per the Bluetooth specification. In the meantime, communication on the first (valid) connection would continue as normal. Allowing only one connection per MAC address is quite simple to implement and, based on our observations during experimentation, appears to be how Windows 7 and Windows XP implement Bluetooth, both of which we were unable to DoS.

VI. Conclusions

We have presented a DoS attack on the connection between a Bluetooth keyboard and a computer which exploits a vulnerability in Windows Vista and Linux BlueZ Bluetooth implementations, allowing us to deny the user functionality of a specific device. This attack will show up only as a device malfunction, and leave all other Bluetooth connections to that computer working properly. When the user then tries to reconnect, the attacker launches a man-in-the-middle attack that gives the attacker full access to whatever the victim types on their keyboard, without the victim being aware of this interception. We have also suggested possible ways to fix this security vulnerability in a simple and easily distributed manner. Future work would include testing other operating systems and different devices to pinpoint the exact location of the security hole, and to implement a patch to the vulnerability we uncovered.

References

[1] Bluetooth SIG, “History of Bluetooth Technology,” [Online] Available: http://www.bluetooth.com/Bluetooth/SIG/History_of_the_SIG.htm.

[2] K. Haataja, “Security Threats and Countermeasures in Bluetooth-Enabled Systems,” PhD Thesis, University of Kuopio, Finland, 2009.

[3] Y. Shaked, and A. Wool, “Cracking the Bluetooth PIN,” in *3rd International Conference on Mobile Systems, Applications and Services*, 2005, pp. 39-50.

[4] C. Franklin and J. Layton, "How Bluetooth Works", [Online] Available: http://electronics.howstuffworks.com/bluetooth2.htm

[5] M. Jakobsson and S. Wetzel, “Security Weaknesses in Bluetooth,” in Topics in Cryptology – CT-RSA, 2001, pp. 176-191.

[6] S. Janssens, “Preliminary Study: Bluetooth Security,” [Online] Available: http://drdeath.myftp.org:881/books/Blueto oth_security.pdf, 2005.

[7] F. L. Wong, F. Stajano, and J. Clulow, “Repairing the Bluetooth pairing protocol”, in *Proceedings of 13th International Workshop on Security Protocols*, Cambridge, England, 20-22 Apr 2005

[8] L. Caldwell, S. Ekerfelt, A. Hornung, and J. Y. Wu, “The Art of Bluedentistry:

Current Security and Privacy Issues with Bluetooth Devices," unpublished, Dec. 13, 2006.

[9] R. Chang, and V. Shmatikov, "Formal Analysis of Authentication in Bluetooth Device Pairing," *FCS-ARSPA*, 2007.

[10] D. Spill, and A Bittau, "BlueSniff: Eve Meets Alice and Bluetooth," *USENIX WOOT*, 2007.

[11] M. Moser, "Busting the Bluetooth Myth – Getting RAW Access," [Online] Available: www.remote-exploit.org/research/busting_bluetooth_myth.pdf.

[12] A. Lindell, "Attacks on the Pairing Protocol of Bluetooth 2.1," *Black Hat USA*, 2008.

[13] J. Jersin, and J. Wheeler, "Security Analysis of Bluetooth 2.0 + EDR Pairing Authentication Protocol," unpublished, 2008.

[14] K. Scarfone and J. Padgette, "Guide to Bluetooth Security," NIST SP800-121, 2008.

[15] W. Ma, A. Mbugua, and D. Poon, "Keystroke Logging of a Wireless Keyboard," unpublished, 2007.

[16] O. Whitehouse, "War Nibbling: Bluetooth Insecurity," @stake, Research Report, 2003.

[17] "Bluetooth Specification Documents," [Online] Available: http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/.