MITM: Logging a Bluetooth Keyboard

Kimberly Paterson
Cal Poly San Luis Obispo
klpaters@calpoly.edu

Ryan Verdon
Cal Poly San Luis Obispo
rverdon@calpoly.edu

ABSTRACT

Despite the security improvements made to Bluetooth since its debut in the late 1990s, Bluetooth technology still has many unresolved vulnerabilities that allow for attackers to execute successful Man in the Middle (MITM) attacks. In particular, the addition of Secure Simple Pairing (SSP) in version 2.1 introduced more security flaws, despite it's purpose to secure the pairing process between devices. This work focuses on launching a successful MITM attack against a Bluetooth-enabled keyboard and a victim PC and logging the data transmitted from the keyboard to the victim PC. Although there is much work that has been done on MITM attacks and attacking PIN vulnerabilities, there has been no published research on successfully logging the keystrokes from a Bluetooth keyboard using a MITM attack.

Categories and Subject Descriptors

H.4 [**Bluetooth Attacks**]: Security

General Terms

MITM Attack, Bluetooth Security

Keywords

MiTM, Bluetooth Attack, Keylogging

1. INTRODUCTION

Bluetooth technology has proven to be vulnerable since it's adoption in to mainstream devices [17], and while the security continues to improve, the threat for Man in the Middle (MITM) attacks against two paired Bluetooth devices remains high [16]. In particular, MITM attacks take advantage of the Secure Simple Pairing (SSP) security protocol that was implemented in version 2.1 of the Bluetooth stack during 2007, which was designed to enhance the security of Bluetooth devices and to simplify the pairing process for end-users. However, our work, and much of the previous work that has focused on SSP MITM attacks, strives to show that there are still inherent security flaws in the implementation of the Bluetooth protocol.

Several researchers have focused on what to do with a MITM Bluetooth attack, particularly when dealing with input devices such as Bluetooth mice [4] and Bluetooth keyboards [9, 14]. However, no work has successfully implemented a Bluetooth keylogger that takes advantage of a MITM attack and that does not require physical access to the victim's devices. Our work creates a proof of concept for a Bluetooth keylogger that can be executed using off-the-shelf hardware.

The rest of the paper is structured as follows. We discuss the background of Bluetooth and Bluetooth security in Section 2. In Section 3, we present the related work in the field regarding SSP vulnerabilities. In Section 4, we present our threat model. Section 5 covers the details of our attack model and implementation options. Section 6 covers the particular implementation and execution of our attack. In Section 7, we present our results and some ideas for how to mitigate any MITM attacks against Bluetooth keyboards in Section 8. Lastly, we discuss future work and our conclusions in Sections 10 and 11, respectively.

2. BACKGROUND

This section discusses the background of Bluetooth and several security features Bluetooth uses.

2.1 Bluetooth

Bluetooth is a short-range communications technology that is used in a wide variety of devices [3]. The types of devices that use Bluetooth technology vary from keyboards to fully functional computers [3]. When two Bluetooth enabled devices are nearby one another, they can perform a pairing process to connect and enable data transfer [3]. Once the devices are connected, they form a piconet and can communicate wirelessly [3]. Any single Bluetooth device can connect with seven piconets at one time and inside any given piconet there can be up to seven Bluetooth devices connected [3].

2.2 Bluetooth Security

Bluetooth implements a variety of security measures to protect itself. The following list are the major security features implemented in the latest version of Bluetooth:

Frequency hopping: The Bluetooth protocol cycles through a wide range of frequencies to avoid conflicts and collisions with other Bluetooth devices [14]. While not a security feature, frequency cycling make it harder to capture packets from a remote host without having special hardware to scan the entire frequency spectrum. However, special hardware like the Ubertooth dongle can map out the full spectrum and remotely sniff packets.

Discoverable modes: Bluetooth devices can choose to only broadcast their availability and Bluetooth address for limited durations [14]. Hiding the device limits the risk of other devices discovering it and trying to connect to it, although devices can remain always discoverable. Additionally, attackers can still obtain the Bluetooth address by performing a brute force attack on the address space [14].

PIN authentication: To pair two Bluetooth devices together there are several authentication options. One of the more secure options is when a PIN is used on both devices to authenticate the devices that the user is pairing [14]. Not all modes require two-way authentication; there exists a mode where no-PIN is needed or a default or static pin is used, thereby removing any security protections. Also, even if a PIN is used, it is only numeric characters which makes it easy to crack [14, 18, 13].

Security Modes: To deal with back-compatibility as well as less-functional devices such as Bluetooth receivers, Bluetooth implements 4 security modes with varying levels of security [14]. For backwards compatibility, the lowest level of security can still be used.

1. Mode 1: no security or encryption
2. Mode 2: no security until channel is established on L2CAP level (software)
3. Mode 3: has security before link setup on the LMP level (hardware)
4. Mode 4: introduced Secure Simple Pairing, pairing protocol to protect devices while a pairing is completed

Most PCs use Mode 4, which enables the SSP process and provides encryption; however, peripheral devices like keyboards and mice, which support Mode 4, can still connect to PCs and use Mode 1 for transferring data. This exposes the data to anyone sniffing the traffic.

2.3 Secure Simple Pairing (SSP)

Secure Simple Pairing (SSP) was introduced in revision 2.1 and has remained largely unchanged in the subsequent versions, although some of the authentication protocols have changed [6]. The original purpose of SSP was to simplify the pairing process for the end-user and to improve security, especially against MITM attacks [16]. In order to simplify the process, 4 different association modes were created, and are used depending on the capabilities of the devices. The 4 association models are as follows:

Numeric Comparison: a 6 digit number is displayed on both devices and a user confirms that the numbers match (this is designed for devices that both have display and input capabilities)

Passkey Entry: designed for when one device has input (keyboard) and the other has display capabilities; the device that has the display shows a pin number, which the user types in on the input device.

Just Works: designed for when one device has no input or display capabilities; the user must accept the connection without being able to visually confirm that the pin numbers are the same

Out of Band (OOB): designed to connect when two devices are touched together, (e.g. Near Field Communication or NFC)

Passkey Entry mode is the primary mode of pairing keyboards and computers; however, keyboards can connect in *Just Works* mode, which bypasses the security features of SSP [12].

2.4 MITM Bluetooth Attacks

We focus on MITM attacks against devices that use Passkey Entry as their primary mode of authentication for pairing. Historically, these attacks have been simple to implement and successful using a variety of approaches.

Among the vulnerabilities the National Institute of Standards and Technology (NIST) lists for Bluetooth versions before 4.0 is the weakness of one-way-only challenge and response authentication used in SSP [17]. Since Bluetooth devices often do not have display or input capabilities, *Passkey Entry* or *Just Works* modes for SSP are necessary to connect less-functional devices, such as keyboards, to more functional devices such as computers or tablets. When only one device is asking for authentication, it's possible to spoof the keyboard into pairing into *Just Works* mode and impersonate the keyboard to the victim computer. Despite NIST's statement that the one-way-only challenge affects versions before 4.0, if an attacker can force a keyboard into *Just Works* mode or use social-engineering or brute-force pin attacks in *Passkey Entry* mode, the device becomes vulnerable to MITM attacks.

2.4.1 Pin Vulnerabilities

NIST recommends that users use a 6-digit random pin when authenticating either in Numeric Comparison or Passkey Entry mode, however, this feature is not mandated. Some operating systems may allow for fewer than 6-digit pins, which are less secure, or may let users enter custom pins, which will not be random and may be less secure. Assuming developers follow the recommended protocol, the probability of guessing the pin is one in a million according to NIST [17].

NIST also claims that since the 6-digit passkey is not incorporated into the link key generation, it isn't useful for an eavesdropper to use. Nonetheless, Lindell showed in 2008 that the pin becomes useful to an attacker when attempting to pair with a device when the passkey is used more than once [13]. Knowing the pin can allow an attacker to connect with the device when the victim attempts to connect his/her devices again. It is possible to force a user to re-pair their devices either by a Denial of Service (DoS) attack [9] or by making the victims assume that the victim devices forgot the link key[18]. When the devices try to pair again, an attacker can intercept the process and authenticate itself to each device.

If the attacker does not know the pin, he/she can, in fact, crack it as Shaked and Wool showed in 2005 when they were able to crack a 4-digit pin in 0.27 seconds by observing Bluetooth traffic [18]. Similarly, in 2012, Barnickel, Wang, and Meyer were able to able to execute a MITM attack by first jamming the connection between the connected devices, calculate the passkey and then re-pairing with the devices [6]. Both of these works and Lindell's work exploit the vulnerabilities of the *Passkey Entry* protocol, where the bit-by-bit evaluation of the passkey by devices can be used to leak the passkey a bit at a time, to determine the passkey quickly. As these works show, computing the pin is alarmingly simple and fairly useful when pins are reused; currently, this is still a feasible attack, but is less likely when pins are not fixed (such as with headsets or other fixed-pin devices).

2.4.2 Keylogging

Regardless of how infeasible it is to guess the password when it is not being reused, some successful approaches target *Passkey Entry* mode since attackers can force a user to enter the passkey for them. This is especially useful for devices that rely on one-way authentication such as Bluetooth keyboards, which do not necessarily have display capabilities. Typically, Bluetooth enabled keyboards will connect to a device with display capabilities (e.g. a tablet or PC) and the device will display a passkey that the user must input through the connected keyboard. If an attacker can trick the user into entering the passkey for him/her, they sidestep the pin guessing process and the attack becomes quite easy.

The benefit of attacking a keyboard in this way is having direct access to the data being imputed into the keyboard without the need for physical access to the device or the software on either the keyboard or device it's connected to. Additionally, if an attack can successfully execute a MITM attack by logging the keyboard input and then forwarding the data to the victim PC or tablet, the user may not even be aware that their data are compromised.

Keylogging with a MITM attack is theoretically possible and some work has been done toward a seamless attack of this sort. In 2007, Ma, Mbugua, and Poon were able to use a pin attack against a paired computer and keyboard by sniffing the Bluetooth traffic and cracking the pin [14]. However, they concluded that keylogging was infeasible because of the cost. A more successful approach was done by Cuthbert et al. where they first performed a DoS attack to disconnect the PC from the keyboard, and then tricked the keyboard into pairing with their attack PC in unencrypted mode [9]. The attack PC then sends a link request to the victim PC, which connects through the first few stages of authentication and then displays a pin for the user to enter. Thinking that this is a malfunction of the connection, the user unwittingly types the passkey to the attack PC, which then enters the pin into the victim PC. Although Cuthbert et al. were able to connect to both devices, they did not execute a successful keylogger, although they argued it would not have taken much more work.

Our work is based on this last approach, but we implement a successful keylogger that forwards the user's data to the victim device.

3. RELATED WORK

MITM attacks are commonly researched for Bluetooth technologies and there are a variety of approaches to executing the attacks. As previously mentioned, the PIN prevents potential issues and is vulnerable to social engineering attacks. Phan and Mingard show that a malicious device can impersonate a legitimate device and communicate with another legitimate device by making the user enter whichever authentication pin it chooses [15]. Other approaches target reused pins by eavesdropping on the pairing process, jamming the signal, and then attempting to pair to the devices using the already-used pin [6, 13]. These approaches rely on the victim not thinking critically about the pin – in some cases, the pin is hard-coded and so must be reused, or is set to default, or the user will use the same pin out of convenience. For this reason, these attacks rely on user-error, which is a largely used resource for MITM exploits.

Another interesting approach is the BT-Nino-MITM attack or Bluetooth No Input, No Output attack [12] . In this attack, the attacker forces the victim device into *Just Works* mode, which does not provide any encryption or authentication. The attacker then jams the physical layer by hopping frequencies along with the victim and sends fake data. The authors also propose jamming the entire 2.4GHz frequency to for the user to try *Just Works* mode.

4. THREAT MODEL

In our threat model, we are assuming the attack will be carried out in a public place where the attacker can get within 30-100 feet of the victim. Examples include places like an airport, an office building or a coffee shop. In any of these places, it would be extremely easy for an attacker to get nearby the victim without appearing suspicious.

The victim of the attack is someone who is using a Bluetooth keyboard to type on a device. The devices could include a tablet, laptop, or desktop PC. The victim would eventually type some sensitive information. This could include payment information, passwords, or some other financial information.

The person performing the attack will most likely be using a laptop. The attacker may or may not have speciality sniffing hardware. His or her main goal will be attempting to log passwords or financial information like credit card numbers. To perform the attack, the attacker would most likely sit inconspicuously nearby the victim or hide out in a van outside the public place where the attack is being carried out.

5. ATTACK OVERVIEW

Based on prior research, the theoretical attack would take 4 main steps to execute, which are as follow:

1. Find Bluetooth Address of the keyboard

2. Execute Denial of Service attack against victim
3. Pair with keyboard and victim
4. Forward keystrokes to victim

Each step is outlined in detail in the following subsections.

5.1 Finding the Keyboard Bluetooth Address

Obtaining a device's Bluetooth address is quite simple. If the device is in discoverable mode, then the address is easily found using a Bluetooth scanner or command line tool such as `hcitool scan` from the BlueZ Linux utilities. If the device is not in discoverable mode but is connected to another Bluetooth-enabled device, then the address can be found using a Bluetooth sniffer and packet capturing software such as Kismet[2]. Project Ubertooth [1] provides both hardware and a plugin for Kismet that allows you to promiscuously sniff Bluetooth packets, and displays the Bluetooth address from whatever device is sending out the packets. Without pairing to the device, it is possible to sniff the packets from nearby devices, although the data is likely encrypted [10].

In order to execute the attack, the attacker will need to know the Bluetooth address for both the keyboard and the victim PC. If the attacker does not have access to hardware like Ubertooth One, he or she can obtain the Bluetooth address after performing a DoS attack, at which point the user may make the devices discoverable in order to reconnect the devices and the attacker can scan for the device addresses.

5.2 Denial of Service Attack

Denial of Service (DoS) attacks are powerful attacks that can disrupt device function or cause harm to the device [**?**]. The purpose of the DoS attack with our approach is to trick the user into initiating pairing. Although devices, once paired, do not technically need to be re-paired, a user who is frustrated with Bluetooth not connecting may try to pair the devices again to re-initiate the connection. Ultimately, this means the devices become discoverable to the attacker, who swiftly swoops in to pair with both devices.

There are several known approaches to executing a DoS attack against Bluetooth, present a realistic and serious problem for Bluetooth users [5]; an attacker can force some operating systems to disconnect with a keyboard by spamming the victim PC with link requests [9]. This attack worked for Windows Vista and Linux Ubuntu with BlueZ but did not work for Windows 7; as a result, if the attacker does not know what OS the user has installed, this approach may not be effective.

Another way to perform DoS is to jams the frequency the device is operating on in order to force the user into re-pairing. The attacker can jam the Bluetooth 2.4GHz frequency or frequency hops with the victim and sends fake data [12]. Once the attacker jams the signal, he or she can attempt to connect using a reused pin[6], which is effective against poor random pin generation or user-generated pins that some operating systems implement.

5.3 Getting in the Middle

What makes this attack possible is the unreliability of Bluetooth connections. Users often experience problems connecting devices and understand that disconnections between their devices are common, so service interruptions are not out of the ordinary and are not suspicious. To fix connections, as in the case of a DoS attack, users will often try to re-pair their device or toggle the physical connect switch or button to try and connect the devices again. Once the user makes his or her devices discoverable, the attacker can quickly pair with both devices. The process must be executed quickly so that the victim keyboard does not have time to connect to the victim PC.

The first step is to pair with the victim keyboard and connect in unencrypted mode, or security Mode 1. This means all data coming from the keyboard will not be encrypted and can be easily dumped and extracted later. If the attacker is successful in pairing with the keyboard and connecting in unencrypted mode, the keyboard is no longer discoverable, which means it will not show up as ”available” on the victim PC's Bluetooth device manager.

The second step is to change the BlueZ configuration file to mimic the appearance of the user's keyboard. When the keyboard becomes discoverable, `hcitool scan` can give you the name of the device, which the attacker simply can change in the Bluetooth config file (found in `/etc/bluetooth/main.conf` for Ubunutu 12.04). The attacker must also change the device class so that the device will show up as a keyboard in the device manager of the victim PC. Typically this device id is `0x000540` for keyboards.

At this point, the attacker relies on the victim thinking the attacker's ”keyboard” is his or her own device (since the victim may not see the actual keyboard in the device manager any longer). The victim, thinking the keyboard is familiar, initiates a pair request to the attacker's PC. Typically, the victim PC will now prompt for a pin from the attacker, but since the user thinks the victim PC is connecting to the victim keyboard, the user enters the pin on the Bluetooth keyboard, which is then forwarded from the attacker to the victim PC, completing the pairing process.

5.4 Forwarding

Although the attacker has now positioned him or herself between the keyboard and the victim PC, the attacker must effectively forward the keystrokes coming from the keyboard to the victim PC, otherwise the jig will be up. Theoretically, it's possible to capture the Bluetooth packets from the keyboard, alter the source address to be the attacker's PC Bluetooth address and then forward the packets using C. However, we were unable to parse the Bluetooth packets because of the implementation differences in libpcap between Bluetooth and Ethernet traffic.

We were able to find a work-around considering that our attacker PC is simply emulating a keyboard; therefore, we looked for HID clients that mimic keyboards and use Bluetooth. We found an Linux tool called hidclient [11] which can essentially forward from any input device on the attacker PC.

Hidclient allows the user to forward from any input device

Key Value	hcidump Encoding
A	0x04
B	0x05
C	0x06
D	0x07
E	0x08
F	0x09
G	0x0A
H	0x0B
I	0x0C
J	0x0D
K	0x0E
L	0x0F
M	0x10
N	0x11
O	0x12
P	0x13
Q	0x14
R	0x15
S	0x16
T	0x17
U	0x18
V	0x19
W	0x1A
X	0x1B
Y	0x1C
Z	0x1D
1	0x1E
2	0x1F
3	0x20
4	0x21
5	0x22
6	0x23
7	0x24
8	0x25
9	0x26

Table 1: Keys to Encoded Value from `hcidump`

the user specifies and that the computer recognizes as a valid input. Since the victim's Bluetooth keyboard is connected to the attacker already, hidclient recognizes the keyboard as an input source. This means the attacker's keyboard is free for typing and will not appear on the victim's screen. The keystrokes are forwarded on to the victim with minimal delay.

5.5 Keylogging

In order to get the data from the Bluetooth packets, we needed to dump the data in a way that it was retrievable either during the attack or after the fact. We considered several packet logger programs including Kismet [2], Wireshark [8], and Mac OSX's Bluetooth Packet Logger. There were a few concerns with these tools: Kismet did not have support without the Ubertooth hardware, and so we did not choose to use this approach; Wireshark has Bluetooth support, but the data is difficult to extract, except through log files; and the OSX Packet Logger was only available on OSX, which meant we were unable to use Linux BlueZ utilities.

```
ACL data: handle 11 flags 0x02 dlen 14
L2CAP(d): cid 0x0041 len 10 [psm 0]
A1 01 00 00 04 00 00 00 00 00
```

Figure 1: Sample `hcidump` dump data (0x04) showing capital A

A simple solution is to use a BlueZ dumping tool called `hcidump` which provided the option to dump the data in hexadecimal form. Since the data coming from the keyboard was unencrypted, this data could be easily extracted from the dump during or after the attack.

To extract the data, we wrote a Python script based on findings of a Chinese blogger, aliased Chopper [7]. Chopper noted that data coming out of `hcidump` was not encrypted but encoded such that characters on the keyboard were given hexadecimal values that did not correspond to their ASCII values, as shown in Figure 1. However, decoding the output proved quite simple, and can be determined by simply pressing a key and looking at the corresponding `hcidump` output. We decoded some of them in our Python script to show a proof of concept, the values of which are shown in Table 1.

6. EXECUTING THE ATTACK

To execute this attack, we used completely off-the-shelf hardware and software. The only custom software and hardware would be the incorporation of the Ubertooth device and plugin for Kismet.

6.1 Hardware

For this attack, we wanted to test several versions of Bluetooth as well as different Operating Systems to determine how effective the attack is against different hardware and software. The following are the devices we used for the attack:

Keyboard: we used a basic Bluetooth keyboard that uses Bluetooth V3.0+EDR (Enhanced Data Rate (EDR), an optional speed-up of data transfer)
Attack Device: we used a Lenovo Thinkpad z61t laptop running Ubuntu 12.04 with Bluetooth 2.0+EDR
Victim Devices: we attacked a mid-2012 Apple Macbook Air running Ubuntu 12.10 via VirtualBox with Bluetooth 4.0 card and a desktop PC with a Bluetooth dongle, running Ubuntu 12.04 and Bluetooth 4.0.

To execute pairing, the Macbook, the Thinkpad Laptop and Tablet all use *Passkey Entry* authentication, where a pin must be entered into the keyboard. Since the attack relies on SSP, which is largely the same across all versions of Bluetooth since it's implementation in version 2.1, the differences in Bluetooth versions across the devices should not matter. The largest variable in whether or not the attack is possible is how the OS handles the DoS attack.

6.2 Setup

Our attack setup used two laptops mentioned in the Hardware section and the Linux Command line utilities mentioned in the previous section. We used the Lenovo laptop as the attacker PC and the Macbook Air as the victim. Since the attack is theoretically OS ambivalent, the attack could theoretically be executed on a variety of Bluetooth enabled devices. However, given out limited resources, we were only able to attack the Macbook Air and another Desktop running Ubuntu 12.04 successfully.

6.3 Implementation

Our worked under the assumption that the pairing process was performed successfully (as prior work suggests is possible) and the attacker is ready to connect and log the keystroke data. To being the keylogging attack, the attacker first scans for the Bluetooth address of both the victim PC and the keyboard by using `hcitool scan`. Once both the address are located, the attacker pairs with both and connects to the devices to begin data transfer.

To connect to the keyboard in unencrypted mode, the attacker uses the following command, where the <keyboard-address> is the Bluetooth address of the victim keyboard:

```
sudo hidd -connect <keyboard-address>
```

Once the attacker has connected to the keyboard, he or she must run the `hcidump` in another shell and dumps the output into a log file, using the following command:

```
sudo hcidump -x > <log-file>
```

The `-x` option dumps the data in hexadecimal encoding, which is easily extracted later.

The attacker then runs the `hidclient` with the `-l` option, which lists the available inputs that can be forwarded to the victim PC:

```
sudo hidclient -l
```

The Bluetooth keyboard appears as one of the options from the `-l` output, and the attacker must simply replace the X in the following command to specify the input source as the keyboard.

```
sudo ./hidclient -eX -x
```

At this point, the victim connects to the attacker as keyboard input, completing the connection. The `hidclient` forwards the data from the keyboard which is being logged by the `hcidump` on the attacker's machine. As mentioned before, this attack relies on the victim being tricked into pairing and connecting to the attacker's PC. For the purposes of this attack, we connected from the victim's PC using the `hidd` command in Linux. In a real-world attack, the victim should be able to connect to the attacker "keyboard" using the OS's built-in device manager.

The attack continues until the attacker or victim severs the connection, at which point the attacker runs a simple script to extract the data. We used a simple Python script to decode the output with excellent success.

7. RESULTS

We successfully created a proof of concept implementation that can forward keystrokes from a Bluetooth keyboard through the attacker to a victim machine. The proof of concept currently only works in the ideal conditions. However, this approach is superior to simple Bluetooth packet sniffing, because it ensures that the data being read from the keyboard is unencrypted, which is not usually the case for most paired keyboards.

8. MITIGATION

We propose three countermeasures to prevent against Man in the Middle attacks on Bluetooth keyboards:

1. Keep devices out of discoverable mode as much as possible. This way an attacker would need special hardware to get the Bluetooth address and it helps prevent the device from becoming a target.

2. Keyboards should require a PIN to pair every time and the PIN should be at least 6 digits and random since keyboards can potentially handle much more sensitive data than a device such as a Bluetooth receiver.

3. Bluetooth manager applications should be smarter. If they notice a previously paired keyboard or device with the same name that has a different Bluetooth address that might warrant a red flag to be displayed to the user.

9. FUTURE WORK

We wish to improve upon this attack with the following augmentations:

1. Implement the entire attack. Including the DoS and pairing without the user's knowledge.

2. Purchase Ubertooth and successfully use it to choose victims without having to have prior knowledge about the victim and the keyboard.

3. Implement the entire attack in a single C/C++ application. This would make testing and debugging much easier to perform. Instead of requiring several separate tools combined together.

4. Test with real-world situations and see if we can acquire passwords.

10. CONCLUSIONS

Our work shows that a Bluetooth keylogger is possible to do. We successfully forwarded packets from the keyboard to the victim with no delay. However, accomplishing this task was easier said than done. We had to use several software packages in the correct order to get the behaviours we wanted. The main conclusion we have is that Bluetooth is needlessly complex. The API is horrible and underdeveloped and warrants much better documentation. The documentation was confusing and non-comprehensive and led us in the wrong direction much of the time. If anyone could improve Bluetooth in any way we recommend making the API cleaner and standardized and improving the documentation that goes with it.

11. REFERENCES

[1] Project ubertooth, 2011.

[2] Kismet, dec 2012.

[3] Bluetooth basics, mar 2013.

[4] Bluetooth mice can leak your passwords!, 2013.

[5] S. Bandyopadhyay, W. Adi, T. Kim, and Y. Xiao. *Information Security and Assurance: 4th International Conference, ISA 2010, Miyazaki, Japan, June 23-25, 2010, Proceedings*. Communications in Computer and Information Science. Springer, 2010.

[6] J. Barnickel, J. Wang, and U. Meyer. Implementing an attack on bluetooth 2.1+ secure simple pairing in passkey entry mode. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 17–24. IEEE, 2012.

[7] Chopper. Analysing bluetooth keyboard traffic with hcidump, jun 2010.

[8] G. Combs. Wireshark the world's foremost protocol analyzer.

[9] T. Cuthbert, A. Gontarek, E. Jensen, and P. Robbins. A bluetooth keyboard attack.

[10] Hak5Darren. Hak5: Capture and analyze bluetooth packets with kismet, wireshark and the ubertooth one in backtrack 5, jul 2011.

[11] A. M. Hoffmeister. hidclient, virtual bluetoothÂő keyboard and mouse, jun.

[12] K. Hypponen and K. Haataja. man-in-the-middle attack on bluetooth secure simple pairing. In *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, pages 1 –5, sept. 2007.

[13] A. Y. Lindell. Attacks on the pairing protocol of bluetooth v2. *Black Hat*, 2008.

[14] W. Ma, A. Mbugua, and D. Poon. Keystroke logging of a wireless keyboard, 2007.

[15] R. C.-W. Phan and P. Mingard. Analyzing the secure simple pairing in bluetooth v4. 0. *Wireless Personal Communications*, pages 1–19, 2012.

[16] S. N. Premnath and S. K. Kasera. Battery-draining-denial-of-service attack on bluetooth devices. *Thanks to*, page 3, 2008.

[17] S. Sandhya and K. Devi. Analysis of bluetooth threats and v4. 0 security features. In *Computing, Communication and Applications (ICCCA), 2012 International Conference on*, pages 1–4. IEEE, 2012.

[18] K. Scarfone and J. Padgette. Guide to bluetooth security. *NIST Special Publication*, 800:47, 2012.

[19] Y. Shaked and A. Wool. Cracking the bluetooth pin. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 39–50. ACM, 2005.