BlueSniff: Eve meets Alice and Bluetooth

Dominic Spill
University College London
d.spill@cs.ucl.ac.uk

Andrea Bittau
University College London
a.bittau@cs.ucl.ac.uk

Abstract

Much of Bluetooth's data remains confidential in practice due to the difficulty of eavesdropping it. We present mechanisms for doing so, therefore eliminating the data confidentiality properties of the protocol. As an additional security measure, devices often operate in "undiscoverable mode" in order to hide their identity and provide access control. We show how the full MAC address of such master devices can be obtained, therefore bypassing the access control of this feature. Our work results in the first open-source Bluetooth sniffer.

1 Introduction

Bluetooth is a widespread technology used in many devices which receive and transmit confidential data. For example virtually all modern mobile phones are Bluetooth capable, enabling them to communicate with a headset or a computer. The data transmitted over Bluetooth is often sensitive, such as voice data or private files, hence making the confidentiality of Bluetooth packets a relevant matter. If it were possible to eavesdrop on Bluetooth, attackers could for example have the ability to intercept address books when they are being synchronized between a phone and computer, thus breaking all of the confidentiality requirements of these devices.

To provide privacy, Bluetooth supports optional encryption at the link layer. This scheme has been shown to be vulnerable if the attacker is able to eavesdrop the paring procedure, which is necessary before two devices can setup an encrypted link [14]. Thus both encrypted and unencrypted links would be threatened by attackers with eavesdropping capabilities. The attack on Bluetooth's encryption was not implemented in practice due to the lack of mechanism for eavesdropping the protocol. Monitoring a Bluetooth connection is not a trivial task and various aspects of the protocol implicitly add to its security and difficulty of eavesdropping.

There are two main hurdles to overcome when attempting to eavesdrop Bluetooth: frequency hopping and data whitening. Bluetooth sends each packet on a different frequency and hops 1,600 times a second. The hopping sequence is unknown to an attacker therefore making it impossible for him to follow an entire conversation, unless the whole spectrum of 79 channels is being monitored concurrently, which is impractical. The second hurdle is that data is whitened (scrambled) making it impossible for the attacker to inspect the payload, or indeed the Bluetooth headers. We show that it is possible to determine the parameters necessary for calculating the hopping sequence and unwhitening data, thus providing a practical mechanism for eavesdropping Bluetooth. We have not yet implemented channel hopping due to hardware restrictions although we are able to successfully eavesdrop on a single channel. By bridging the gap between the previously theoretical attack on Bluetooth's encryption, the protocol's data should no longer be regarded as having any practical confidentiality properties.

Recently, "undiscoverable mode" is being widely used as a form of protection in Bluetooth devices. This mode acts in a similar way to a firewall, causing a device to become stealth without ever making it reveal its MAC address. Hence only trusted devices which have prior knowledge of the MAC address would be able to connect to an undiscoverable device. This form of protection is often used in devices with hardcoded PINs (usually 0000) in order to provide access control. A common example of such devices are headsets which can be turned on in a special manner in order to make them discoverable for the initial paring, and then switched on in their normal state of undiscoverable. This protection often is the only option when the alternative would instead be to integrate a display with a keypad for configuring customizable PIN numbers—which can be larger than the device itself.

We provide a mechanism for determining the MAC address of undiscoverable "master" devices, that is, those

that initiate the connection. It is no longer the case that undiscoverable mode can be relied upon as a mechanism for access control and secrecy—the MAC address can be obtained. This is a real threat to users which enable undiscoverable mode in order to "patch" vulnerabilities in their devices. For example, many mobile phones have vulnerabilities ranging from allowing attackers to download arbitrary files to arbitrarily controlling the device [9, 12]. All of these attacks require knowledge of the MAC address so one way to mitigate the problem is to make the device undiscoverable. Unfortunately this is commonly used and encouraged as a technique to avoid unwanted connections [5], but it has been shown that it is still possible to connect to devices in this mode [6] if the MAC address is known.

Our work brings forward two main contributions. First, we show that the Bluetooth packets have no confidentiality properties. Specifically we demonstrate how data can be unwhitened and the hopping sequence calculated. Prior work has shown how the data can be decrypted if necessary [14]. Second, we show that the undiscoverable mode does not provide access control to master devices, nor protects the secrecy of their MAC address. We are able to determine the complete MAC address of these devices. Finally, all our work was done using GNU Radio and we therefore provide the first open-source Bluetooth sniffer, free from any licensing restrictions.

2 Bluetooth

Bluetooth is a wireless protocol operating in the 2.4GHz license-free band. It is often used for low speed data transfer between low power devices, such as voice calls between telephones and wireless headsets. Eavesdropping on Bluetooth packets largely comes down to two variables held within each Bluetooth device, the MAC address and the clock. The MAC address is a unique identifier allocated to the device from the same pool as the MAC addresses for most modern networking hardware. The clock is a 3.2kHz counter stored in a 28 bit number, and it wraps approximately every 23 hours.

Bluetooth uses frequency hopping over 79 channels in order to minimize interference and (usually) hops once every 625μs, sending one packet per channel. The hopping sequence is determined by the MAC address of the master device and its clock. The master device is the one that initiates the connection, and the slave being the one connected to. Data transfer often takes place on alternate hops, leaving the intervening hops for acknowledgments, as shown in Figure 1. The difficulties in eavesdropping in the presence of channel hopping are therefore determining the sequence, and hopping quickly enough.

The second hurdle to eavesdropping on data carried

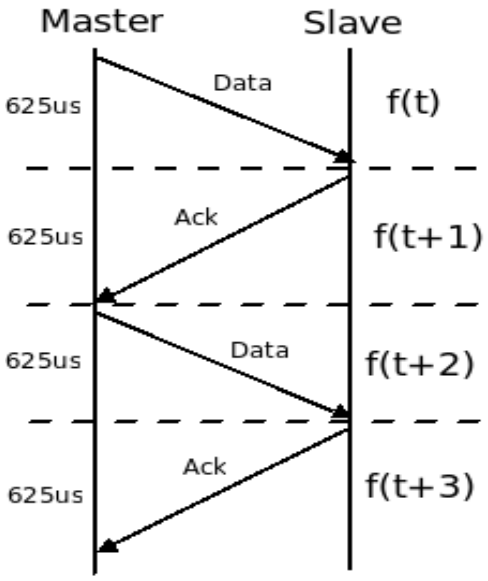

Figure 1: Data transfer is often unidirectional with acknowledgments transmitted in the intervening timeslots. f(t) is the frequency used at time t, calculated from the clock and the MAC address.

in Bluetooth connections is that packets are *whitened* (scrambled) in order to improve error resilience and security. The whitening is determined by six bits of the master device's clock. Thus, in order to eavesdrop on a Bluetooth connection two pieces of information are needed: the MAC address and clock of the master device. With this information, one can calculate the hopping sequence and tune to the correct radio channel, and then unwhiten the received packets.

Most Bluetooth devices offer a mode called "undiscoverable" which prevents a device from advertising its MAC and clock upon receiving an inquiry. This is often used by people for access control as suggested by Bluetooth security advisories [5], so inquiry cannot be relied upon for determining the MAC and clock. The MAC address cannot be easily discovered since Bluetooth packets do not contain the complete address but only the lower three bytes, known as the Lower Address Part (LAP). The clock is not present in standard packets. There is one packet (FHS), used during the connection handshake, that contains all the necessary information (MAC and clock) for listening on a connection. Unfortunately obtaining this packet requires eavesdropping the start of a connection and being tuned on the (unknown) correct frequency, which most likely is not possible in practice. To obtain the clock one can establish a connection to a device, but in order to do that, the MAC address is needed. Thus, much of the difficulty of eavesdropping Bluetooth comes from the secrecy of the MAC address.

3 Eavesdropping

We have implemented a proof of concept Bluetooth sniffer that operates on a single channel [15]. We are currently working on channel hopping which is proving to be difficult due to hardware restrictions. We will however present our initial results about it.

There are three problems to eavesdropping Bluetooth in practice. First, standard Bluetooth dongles do not have a concept of "promiscuous" mode so we need to make use of special hardware such as software defined radio. Second, we need to unwhiten the data since all Bluetooth data is scrambled (whitened) by default. Third, we need to determine the master's MAC address since this allows us to calculate the hopping pattern.

In order to capture packets, we made use of GNU Radio. We then developed a technique for unwhitening data which does not require knowledge of the clock. The use of this technique will leak some bits of the clock as a side-effect, and these are enough for an attacker to whiten future data in case that an active attack needs to be performed. Finally we developed a mechanism for determining the full MAC address of a device after eavesdropping one packet and transmitting less than 256 packets. Having obtained the MAC address we can connect to the device in order to determine its full clock, giving us all of the information needed to follow its hopping pattern. All of these attacks work on undiscoverable devices. We will now describe them in detail in the sections that follow.

3.1 The USRP

In order to eavesdrop Bluetooth packets from an arbitrary connection, a radio device is required and the Universal Software Radio Peripheral (USRP) [4] was chosen. The USRP is the hardware device associated with the GNU Radio project [2], which sets out to create an open source framework for implementing radio devices in software. This software was used to demodulate and process Bluetooth packets that had been received by the USRP hardware. The USRP device consists of a motherboard, which has DACs and ADCs on it, and daughterboards which are able to receive, and sometimes transmit, over a specific band of frequencies. For eavesdropping on Bluetooth packets the daughterboard used is locked to the free 2.48GHz ISM band which Bluetooth devices operate in.

There are two problems to be solved when eavesdropping with the USRP. First, the demodulator parameters need to be set up properly so that Bluetooth data can be recovered on the channel that is being listened on. Second, because a Bluetooth connection frequency hops, the radio must be able to monitor other channels too in order to eavesdrop the entire conversation and not only packets that happen to pass by the single channel on which the USRP happened to be tuned on. There are two approaches to the latter problem. One solution is to eavesdrop all Bluetooth channels in parallel. The other approach is to make the USRP retune in order to follow the hopping sequence. In the following sections we describe how we demodulated Bluetooth using the USRP, how we can monitor multiple channels, and how we plan to implement channel hopping in the future.

3.1.1 Eavesdropping on a single Bluetooth channel

A USRP with a 2.48GHz daughterboard can be tuned to any Bluetooth channel. In order to eavesdrop a packet it is sufficient to stay tuned to a single channel and wait for a packet to fly by. Since the channel hopping rate is so high (1,600 hops/s), waiting for one second is more than enough in order to intercept a packet. However, tuning the radio is not sufficient since the correct demodulator needs to be setup and it must be configured in the correct way. To our knowledge we are the first to eavesdrop Bluetooth using the USRP and in fact much of our research effort was spent in trying to determine the correct parameters to enable the radio to demodulate Bluetooth.

Finding a Bluetooth packet for the first time was not a trivial task. The obvious approach would have been to transfer a large file containing a known pattern and then search for the pattern in the output from the demodulator. This basic approach was not possible since the Bluetooth data is whitened by the device and the whitened version is unknown, so we would not know what we were looking for. Thus we were seeking through a sequence of bits, not knowing which bits to look for, and not even knowing if our demodulator setup was correct. When researching the correct demodulator parameters, looking for packet headers turned out to be unreliable since they are very short and incorrect demodulator settings yielded many bits that seemed like a real packet headers but in reality were just noise.

Fortunately we came across a debug mode of Cambridge Silicon Radio (CSR) based Bluetooth dongles that causes packets to be sent repeatedly on a single channel [8]. This allowed us to eliminate variables such as channel hopping, and since the test packets were not whitened, we knew the exact bit sequence we were looking for. After several attempts we were able to demodulate the packet.

Bluetooth data is modulated using a Gaussian Frequency Shift Keying (GFSK) method, which is not strictly supported by the GNU Radio demodulation tools. There is a Gaussian Minimum Shift Keying (GMSK) demodulator, and this was tuned to allow for the demodulation of Bluetooth packets. GMSK is a variant of GFSK in which the frequency shift used to represent data is kept to

a minimum. The Bluetooth modulation scheme is within this limit, so GMSK demodulation is a drop in replacement for GFSK in this scenario.

The main parameters for the demodulator are the modulation index and the symbol rate given in the form of number of samples per symbol. These were found with the help of the CSR debug packets and our modified version of the GNU Radio software oscilloscope which allowed the samples per symbol parameters to be calculated from a received packet. Our modified oscilloscope allowed file input to be used, which lead to a large sample being cut in size until it left a sample file containing a single packet. The packet was measured to be approximately 400μs in length, with a reported sampling rate of 4,000,000 samples per second. The packet was known to be 366 bits in length, which gave a samples per symbol value of four. The modulation index μ, a measure of the frequency deviation of the unmodulated signal, was derived from the Bluetooth specification [1] which allows a range from 0.28 to 0.35, and we found 0.32 to produce good results. Thus, the necessary parameters for demodulating Bluetooth using GNU Radio's GMSK demodulator are:

- Modulation index μ: 0.32.
- Modulation rate (samples/symbol): 4.

It may be useful to filter the incoming signal to remove noise caused by signals transmitted in adjacent channels or other by other wireless systems operating in the same band. However this is not recommended since all of the signal processing is done in software and it can therefore have a negative impact on the performance of the packet demodulation and parsing processes. The experimentation showed that, for our setup, filters were not needed, although this may not be the case in a less controlled environment, or with a greater distance between the USRP and the transmitting device.

With these parameters, it is possible to eavesdrop Bluetooth on any given channel. We now examine the issue of channel hopping, first by showing how multiple channels can be eavesdropped in parallel, and second how the radio may be retuned in order to follow the hopping sequence.

3.1.2 Eavesdropping on multiple channels at once

The bandwidth of the USRP is approximately 5MHz and it is larger than a single Bluetooth channel. This makes it possible to eavesdrop multiple neighboring channels. Using translating filters, which are part of the GNU Radio set of tools, we were able to reliably receive packets at up to 2MHz on either side of the center frequency. The translating filters change the center frequency of the signal in software, allowing multiple channels to be monitored simultaneously so long as they are within the bandwidth of the receiver, in this case approximately 5MHz. This equates to five channels being eavesdropped with one daughterboard and we were able to practically achieve this, extracting data from packets across five neighboring channels. Each USRP can take two daughterboards, and therefore can eavesdrop up to ten channels with a single USRP device.

It is possible for a connection to use a limited number of channels due to regulatory issues. In fact, the Bluetooth Adaptive Frequency Hopping (AFH) parameter can be used to select the available channels, with a minimum of twenty channels. Thus, with a man in the middle attack, it may be possible to reconfigure an existing connection to use only twenty channels and in this case only two USRPs would be necessary to eavesdrop all of the packets involved. We will investigate this further once we implement transmission and move on to active attacks.

To listen to the entire range of the 79 Bluetooth channels, eight USRP devices are needed. All 79 channels need to be filtered, monitored, and then the packets need to be ordered before the data can be read from them. This greatly increases the processing power required, although the captured data can be processed offline or by a cluster of machines. Ordering packets is necessary when using multiple USRPs because data is buffered in a variable way before being delivered and may therefore result in the packets being received out of order from the multiple devices. The processing of these packets is made easier by our technique of deriving the clock signal from each packet, since it acts as a sequence number and therefore allows for some ordering of the packets to take place. More ordering information may be available from data or headers in layers higher up in the protocol stack. It is a future goal for the GNU Radio project to get rid of delays and buffering, so ordering will no longer be a problem once this is accomplished.

3.1.3 Channel hopping

Bluetooth devices retune their radios 1,600 times per second in order to communicate with each other, but unfortunately tuning at such a rate is not an easy task with the USRP. The 2.48GHz daughterboard is able to retune within 200μs, which is not fast enough to follow a Bluetooth hopping pattern since each time slot is 600μs. Hopping with a tuning delay of 200μs would cause up to one third of each packet to be lost. There are various solutions to this problem depending on the attacker's needs.

The attacker could choose which 200μs of the time slot to miss. For example, if the start of the packet has no sensitive data, then the attacker would retune at the end

of the time slot in order to catch the tail of the packet. Otherwise the retune could occur before the end of the time slot in order to catch the head of the next packet. If the attacker knew that the eavesdropped packets were short and (say) occupied only half a time slot, the retune could happen right after the packet ends, and not at the end of the time slot. This leaves the attacker enough time to retune to the next frequency and successfully catch the start of the next packet. Examples of short packets that contain sensitive information are key presses from a Bluetooth keyboard.

It may be the case that only one direction of the connection holds sensitive data. For example, a file download will have incoming data and outgoing acknowledgments. Bluetooth uses an alternating scheme for receiving and transmitting data. Thus, if only one direction needs to be eavesdropped, the attacker has a whole time slot in order to retune and the USRP is fast enough for this. If the whole connection needs to be eavesdropped, two daughterboards are necessary, and they can be installed on a single USRP. In this case, one daughterboard would be listening to the current time slot while the other one would be retuning to the next time slot. These roles would be then switched and alternated between the two boards throughout the entire connection.

The major practical obstacle to any frequency hopping implementation, and the reason why we still have not produced a working prototype that supports hopping, is the buffering and asynchronous nature of the GNU Radio framework. This introduces a delay which is not constant, and is often large enough to allow multiple packets to be received before the buffered data receives any parsing or processing module. The delay means that the software does not have a chance to synchronize with the hopping of the device which is being attacked by adjusting the clock on packet reception in order to compensate for drift. One of the goals for GNU Radio is to allow more direct flows of data processing through the software, removing the delay in processing, meaning that frequency hopping may be possible with future revisions of GNU Radio. This refactoring is a non-trivial change to the GNU Radio software and hence we did not take the route of doing it ourselves in order to enhance our Bluetooth sniffer.

3.2 Removing data whitening

Packets that Bluetooth devices transmit are "whitened", this means that the data in the header and payload is scrambled before transmission. The scrambling is set by the lower 6 bits of the clock, which are known only to the devices involved in the communication.

Whitening is performed on every packet regardless of higher level protocol. It involves an XOR of a pseudo-random sequence with the packet data. The sequence is produced using a six bit value derived from the clock as input to a linear feedback shift register (LFSR), shown in Figure 2. This means that there are a total of 64 possible starting values for the LFSR, which is a small enough set to bruteforce.

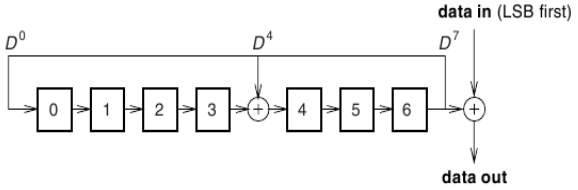

Figure 2: The linear feedback shift register used to generate the whitening data. (Figure from the specification [1].)

Our mechanism is to generate all 64 candidate unwhitened packets and then validate which packet is the correct one using other fields in the packet such as the link ID (typically 1) or the CRC calculated over the payload (which we can check). This process will also reveal six bits of the clock, which can be used to removing whitening from future packets, or to add whitening to a packet in the case that an attacker wishes to transmit data. The clock, once discovered, can be kept synchronized by using the CPU cycle counter as a fine-grained time source.

We now describe how we generate the 64 candidate packets. Each of the 64 possible input values are used in turn to initiate the whitening LFSR, producing 18 bits of data to be XORed with the packet header. We made the process faster by creating a look up table because the whitening LFSR, shown in Figure 2, uses a seven bit register and, as this function can be called up to 102,400 times per second in one of only 127 possible states, we wanted to make this function as efficient as possible. This algorithm is shown below.

```
void unwhiten(input, output, clock) {
 indices[64] = array of indices
               into whitening_data;
 whitening[127] = array of outputs
                  from whitening lfsr;
 index = indices[clock];
 for(bits in data) {
  output = input XOR whitening[index];
  index++;
  index = index MOD 127;
 }
}
```

For each possible value of the clock, one packet header is produced. We now need to determine which one is the

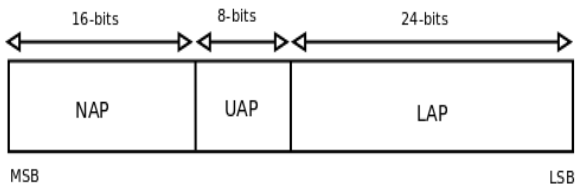

Figure 3: The three parts of a Bluetooth MAC address.

correct candidate. Each header contains connection state, such as the link ID, which is constant throughout the duration of a connection and therefore will be the same in every packet. Over a number of packets these constant values will emerge, allowing us to filter out candates until the correct one is found. Another technique is to look for correlations between packets such as the clock value itself advancing as expected given the packet reception time.

The most practical method for determining the correctly unwhitened packet is checking the payload CRC. With this method, false positive results are extremely unlikely as the CRC is initialized with a byte of zero and the UAP, as is the HEC, so a false positive match would have to match all 16 bits of the CRC initialization (1 in 65,536) and eight of those bits would have to match the HEC initialization. There is one caveat because not all Bluetooth packets carry a payload CRC. However most of the control data sent between devices uses DM1 packet types which do have a CRC, and these packets are very frequent so it is not a problem obtaining such packets in practice.

3.3 Obtaining the MAC

To determine the channel hopping sequence, the master's MAC address and clock need to be known. Of these two parameters only the MAC address is necessary, since with its knowledge, the clock can be determined by establishing a connection with the device. This technique for obtaining the clock will work even if the device is in "undiscoverable" mode [6]. Obtaining the MAC address is also useful for attacking devices which rely on their secrecy for access control, and often this is their only protection mechanism.

The MAC address is divided into three parts called the Lower Address Part (LAP), Upper Address Part (UAP) and Non-significant Address Part (NAP). The lengths of these parts are three, one and two bytes respectively, as shown in Figure 3. In the following sections we show how these parts can be recovered by eavesdropping on any single Bluetooth channel.

3.3.1 Lower Address Part

Determining the lower address part (LAP) is straightforward since it is present in the Bluetooth packet headers. When raw data is received from the GNU Radio, the start of a packet needs to be found. The start of every Bluetooth packet, as shown in Figure 4, has a constant 72 bit pattern, called the access code, and it may be used in order to find packets, it contains the 24 bit LAP along its 34 bit checksum and 14 bits of synchronization and error detection data. Thus the LAP can be simply read from a packet and validated by its checksum. Unfortunately the rest of the MAC address is not transmitted in packets and we need to be more clever about discovering it.

3.3.2 Upper Address Part

The Upper Address Part (UAP) is not stored within each packet, and is only transmitted as part of the handshaking procedure. However, each packet has a header with an error check field which is calculated from the UAP. We noticed that it is possible to reverse this checksum in order to reveal the UAP. The error checking field takes the form of a Header Error Code (HEC), which is calculated over the 10 bits of data in the header, see Figure 4. The register used for this check is initialized with the 8 bits of the UAP.

The HEC calculation is based on a linear feedback shift register (LFSR) which is initialized with the UAP of the master device. Each bit of the header is fed into the LFSR in the order it will be transmitted, and the final contents of the register is appended to the end of the header.

This calculation is based on XOR operations and can be run entirely in reverse, as shown below.

```
UAPtoHEC(header, HEC)  {
  for(bits in header)  {
    HEC = HEC_bit_0 XOR HEC_bit_1;
    HEC = HEC_bit_0 XOR HEC_bit_2;
    HEC = HEC_bit_0 XOR HEC_bit_5;
    HEC = HEC_bit_0 XOR HEC_bit_7;
    right_shift(HEC);
    HEC = HEC_bit_0 XOR header_bit;
  }
  return HEC;
}
```

The LFSR is initialized with the HEC, and each bit of the header is processed in reverse. The final state of the register will be the value with which the LFSR was initialized, *i.e.* the UAP of the master device.

The following is an algorithm and summary for how to determine the UAP and unwhiten data from an eavesdropped packet, this is also shown in Figure 5.

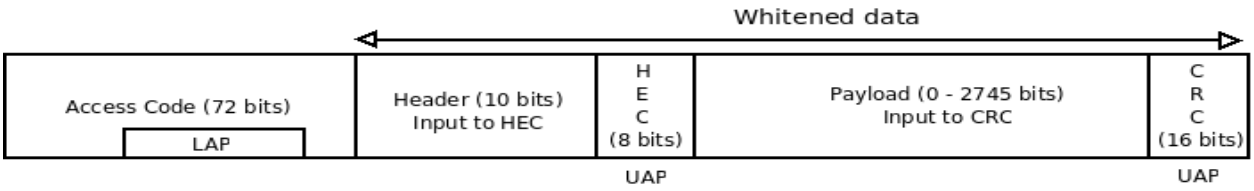

Figure 4: The format of a Bluetooth packet, showing Access Code, Header with HEC and Payload with CRC. The relevant parts of the packet for LAP and UAP extraction are also shown.

1. Generate 64 candidate packets by unwhitening them using all the possible 64 inputs to the whitening LFSR.
2. Reverse the HEC from each packet header and generate a total of 64 corresponding candidate UAPs.
3. For each of the 64 pairs of packet and UAP, calculate the CRC and see if it matches. If so, the correct UAP and unwhitened packet has been found.

3.3.3 Non-significant Address Part

The remaining two bytes of the MAC address are referred to as the Non-significant Address Part (NAP). None of the information in the packet is based on the NAP so there is no way to determine them from received packets. In practice, of the two bytes in the NAP, the first is virtually always zero. Thus, the remaining byte can be bruteforced by sending at most 256 pings to all the possible remaining MAC address combinations.

It is however possible to make educated guesses regarding this byte rather than blindly bruteforcing. The top three bytes of the MAC address (NAP + UAP) correspond to the vendor of the device. MAC address ranges are assigned by the IEEE to vendors in order to divide the namespace so that address collisions are avoided. A complete list of the address ranges is available and known as the Organizationally Unique Identifier (OUI) list [10]. The UAP value may be used to filter out possible candidates from the OUI list. More candidates can be ruled out by guessing the type of device being eavesdropped. For example, when eavesdropping mobile phones, MAC addresses belonging to Nokia are more likely to be correct rather than those allocated to Sun Microsystems. Indeed there are relatively few Bluetooth vendors, and in practice a very narrow set of vendors are widespread. Using this technique usually yields less than thirty candidates, which can be bruteforced very quickly. Instead of using the OUI list, a smaller sample set can be generated by using a database of common Bluetooth device MAC address prefixes [18].

The following algorithm is a summary of how to determine the MAC address from an eavesdropped packet.

1. Determine the LAP from the access code.
2. Use our unwhitening and HEC reversal techniques to obtain the UAP.
3. Use the OUI list to find vendors that end with the UAP and sort them (*e.g.* Nokia and CSR first).
4. Bruteforce the candidate set by sending a ping to the guessed MAC address.

3.4 Determining the Hopping Pattern

In order to calculate the hopping pattern, the MAC address and clock of the master device need to be known. We have shown how to determine the full MAC address. We also demonstrated how our unwhitening reveals 6 bits of the clock, but the rest of the clock, which is necessary for calculating the hopping sequence, remains a mystery. By knowing the MAC address, the solution becomes fairly simple: connecting to a device reveals its clock. The clock of a device is global and not per-connection so leaking the value will enable following any connection from that device. Commodity hardware can be used for establishing the connection and the clock value can be read from the device using tools included in the Linux Bluetooth stack [8]. With this information we are able to calculate the hopping pattern and in principle follow it (hardware permitting).

4 Bringing it all together: eavesdropping an OBEX file transfer

In this section we illustrate how our techniques can be used to carry out an end-to-end attack. Our work is still incomplete and lacks channel hopping so we are unable to eavesdrop on the entire connection, but this example serves as a proof of concept for our eavesdropping using GNU radio, MAC address discovery and unwhitening techniques.

We chose an OBEX [3] file transfer application simply because we can control the payload of the data (file

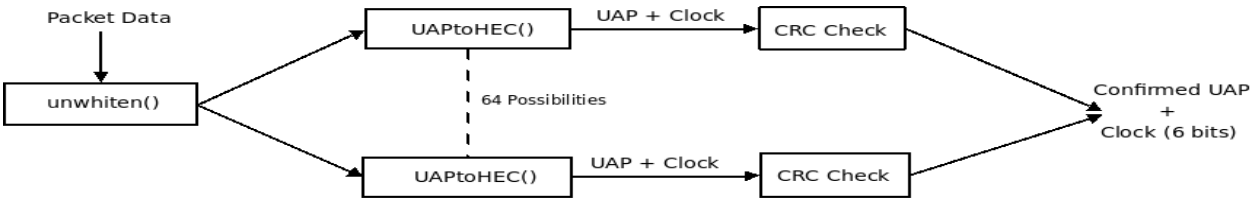

Figure 5: The flow of data through the processes of unwhitening, UAP extraction and UAP confirmation

contents) making it easier to check whether we are doing things correctly. The OBEX data transfer protocol is well established and documented. It has been used for IrDA connections and was chosen as a data transfer method for Bluetooth using the built in RFCOMM transport method as described in volume 7 of the Bluetooth specification [1]. RFCOMM is encapsulated in the Logical Link Control and Adaptation Protocol (L2CAP) [1] to send its data. The L2CAP protocol handles the fragmentation of the data to be transferred, and is transmitted as the payload of Bluetooth packets. The details of these protocols are unimportant, except that they allow a file transfer to take place with the file data easily identifiable in the payload of the packet.

Our setup consisted of two Linux boxes, one acting as an OBEX ftp server and one as an OBEX ftp client. A file containing "101010..." was transferred from one machine to the other while a third box running our GNU Radio module was intercepting the communication. The alternating "101010..." file was used simply because it was easy to find in a large block of output data. We will now explain how we eavesdropped the data, how we determined the MAC address, and how we unwhitened the data and recovered the portion of the file being transmitted.

Firstly the USRP was tuned to a channel that Bluetooth was known to use, in this case 2.421GHz. The choice of frequency is unimportant as Bluetooth attempts to use all channels equally. 2.421GHz was chosen to avoid interference from neighboring 802.11g sources and the CPU clock signal. The software was then able to detect packets being received by the USRP. These were demodulated using the GMSK demodulator from the GNU Radio tools.

4.1 Determining the UAP and unwhitening

The LAP of the packets was extracted from the access code of the first packet found. The UAP was then extracted using the reversed HEC calculation method, giving 64 candidate UAP values. For each of these 64 candidates the rest of the packet was also unwhitened. The CRC could now be checked to identify the correct UAP. The UAP and the six bits of the clock values that are used for whitening were now known. The CPU's cycle counter can be used to maintain the state of the six clock bits so that future packets can be unwhitened without requiring a bruteforce.

The payload format is fairly straight forward for an OBEX transfer, it features the L2CAP header, followed by the RFCOMM header, and finally an OBEX header. Then the data from the file is carried in plain text, the packet ends with a single byte RFCOMM CRC and a two byte packet CRC, as shown in Figure 6. Given this simple packet layout we were able to extract part of the file contents as we eavesdropped packets from the air.

4.2 Determining the full MAC address

So far we revealed the bottom four bytes of the MAC address: 5B:00:FA:C2. Filtering the OUI list for vendor prefixes ending in 5B yields 41 results, meaning that there are 41 candidates to bruteforce in order to reveal the NAP. Pinging a Bluetooth device requires approximately one second. The devices also need to find one another and identify themselves and this takes up to a second because both devices have unique hopping patterns, and these need to coincide on a frequency before communication can take place. Thus the rest of the MAC address can be determined in under two minutes assuming that the first byte is zero (it was the case). The 41 candidates included companies such as CSR and Nokia, and Smart Empire Investments and PanelLink Cinema. An educated guess would attempt CSR and Nokia first before PanelLink, possibly revealing the MAC address instantly.

5 Future directions

Our contributions consisted of opening the door for practical Bluetooth security research, although there is still work to be done. The largest hole in our current implementation is the lack of frequency hopping support. In practice, an attacker would probably use GNU Radio to bootstrap the attack and then feed in the discovered parameters (clock and MAC) to hardware specifically designed for Bluetooth. For example, it may be possible to use $10 Bluetooth dongles and develop a cus-

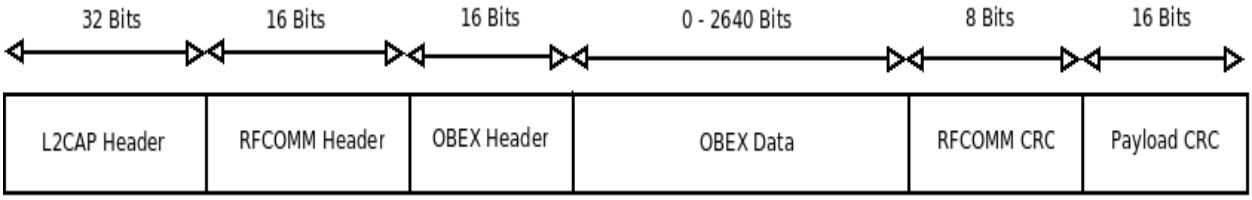

Figure 6: The payload of a Bluetooth packet carrying the start of of an OBEX file transfer.

tom firmware that allows the modification of the MAC address and clock, making this a very economical solution [13]. However, this approach will most likely have legal and licensing problems. We will therefore focus our research on channel hopping with GNU Radio, or more interestingly, in eavesdropping the whole Bluetooth radio spectrum in parallel. This latter approach is of more interest to us since all Bluetooth piconets could be monitored simultaneously. Furthermore, this would be a unique feature to our system since a standard Bluetooth dongle will never have such a capability due to its narrow-band, and it will be at an affordable price with respect to professional wide-band radio receivers.

We intend to investigate transmission and active attacks. With transmission we may be able to reset connections by spoofing a disconnect packet, and therefore examine how tolerable Bluetooth is to man in the middle or DoS attacks. We will also assess the security of the layers above the baseband such as L2CAP, RFCOMM and OBEX to see if they present any vulnerabilities now we have enabled the attacker to have full access to the communication medium.

It is now possible to implement some of the attacks [16, 11, 17, 14] that were only theoretical in nature before, and attacks such as interception of calls that make use of a Bluetooth headset become a real threat. Eavesdropping is the basic weapon an attacker needs and we have provided it; at this point the fun research begins.

6 Related Work

The only available Bluetooth sniffer is FTS4BT [7], a commercial product for eavesdropping Bluetooth in a controlled environment. It is designed as a tool for diagnosing problems and intercepting connections to which the user already had access. In order for FTS4BT to work, the master device needs to be discoverable so that vital parameters such as the MAC and clock can be obtained. These are necessary for unwhitening data and channel hopping. Our sniffer does not require the master to be discoverable, and we are able to unwhiten data even without knowing the clock value. Hence we are able to eavesdrop in arbitrary environments where as FTS4BT is limited to those with discoverable devices, which are now becoming a minority. Because FTS4BT operates on hardware specifically designed for Bluetooth, it is able to channel hop and follow an entire connection. We should be able to match this capability of FTS4BT once the USRP becomes powerful enough. Another distinguishing feature is that FTS4BT costs approximately $10,000 where as a USRP can be bought for ten times less the price. Finally, our solution is open-source enabling other researchers to make use of it without any additional costs and licensing restrictions.

7 Conclusions

We presented techniques for eavesdropping on Bluetooth data, therefore eliminating any confidentiality associated with packets. We show how packets can be intercepted, unwhitened, and prior work has demonstrated how to decrypt data in the case of encryption. We provide the first single channel open-source Bluetooth sniffer. We do not yet support channel hopping due to hardware restrictions although we are able to obtain the necessary parameters for calculating the hopping sequence, thus making it possible to (in principle) eavesdrop an entire connection.

We also show how the full MAC address of master devices can be obtained, therefore bypassing the access control of such devices operating in undiscoverable mode. We have therefore opened the doors for practical Bluetooth security research by providing the means for eavesdropping, and by showing that the common form of security through obscurity (undiscoverable devices) used in Bluetooth is futile.

Acknowledgments

We would like to thank Joshua Lackey, Joshua Wright and Adam Laurie for their invaluable information with regards to Bluetooth hacking. Many thanks to Mark Handley and Brad Karp too for providing us with much feedback on the paper.

References

[1] Bluetooth specification v1.2. http://download.www.techstreet.com/cgi-bin/pdf/free/298250/BT_Core_v1_2%.pdf.

[2] GNU Radio—the gnu software radio. http://www.gnuradio.org.

[3] Obex v1.3 specification. http://irda.affiniscape.com/associations/2494/files/Specifications/OBEX%13_Plus_Errata.zip.

[4] USRP—Universal Software Radio Peripheral. http://www.ettus.com.

[5] F-Secure. Security advice. http://www.f-secure.com/f-secure/pressroom/protected/prot-2-2006/17-407%-3032.shtml.

[6] K. Finisterre. http://www.digitalmunition.com/HijackHeadSet.txt.

[7] Frontline Technology. FTS4BT Bluetooth Protocol Analyzer & Packet Sniffer. http://www.fte.com.

[8] M. Holtmann. Bccmd, part of bluez - the linux bluetooth stack. http://www.bluez.org.

[9] M. Holtmann. BlueSnarf. http://trifinite.org/trifinite_stuff_bluesnarf.html.

[10] IEEE. Organizationally Unique Identifier. http://standards.ieee.org/regauth/oui/oui.txt.

[11] M. Jakobsson and S. Wetzel. Security weaknesses in bluetooth. *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*.

[12] A. Laurie and M. Herfurt. BlueBug. http://trifinite.org/trifinite_stuff_bluebug.html.

[13] M. Moser. Busting The Bluetooth Myth—Getting RAW Access. 2007. http://packetstormsecurity.org/papers/wireless/busting_bluetooth_myth.p%df.

[14] Y. Shaked and A. Wool. Cracking the bluetooth pin. *In the proceedings of the 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, 2005.

[15] D. Spill. BlueSniff. http://www.cs.ucl.ac.uk/staff/a.bittau/gr-bluetooth.tar.gz.

[16] J. Su, K. K. W. Chan, A. G. Miklas, K. Po, A. Akhavan, S. Saroiu, E. de Lara, and A. Goel. A preliminary investigation of worm infections in a bluetooth environment. *In the proceedings of the 4th ACM workshop on Recurring malcode*.

[17] Trifinite group. Trifinite. http://trifinite.org/trifinite_stuff.html.

[18] J. Wright. Bnap Bnap. http://802.15ninja.net/bnapbnap/.