THE AUSTRALIAN NATIONAL UNIVERSITY

School of Computer Science

College of Engineering and Computer Science

COMP3006 - Computer Science Research Project

Bluetooth Sniffing and the PS3

Author:

Luke Vincent

Supervisor:

Eric McCreath

November 1, 2010

Abstract

Bluetooth is a widely deployed networking protocol providing efficient and secure communication for low power-devices. The Sony PlayStation 3^{TM} is a 7th generation gaming conle designed to be controlled wirelessly via Bluetooth controllers. This report first outlines an investigation into the plausibility of sniffing Bluetooth communication of consumer devices. Building on this is an investigation into the implementation of Dominic Spill and Andrea Bittau's research on discovering unknown Bluetooth MAC addresses. Lastly using various penetration techniques I reverse-engineer some of the Bluetooth communication of the Playstation controllers, providing a foundation for an open-source emulation of the controller - the first time this has been achieved in this manner.

Declaration

Except where otherwise indicated, this thesis is my own original work.

Acknowledgements

I acknowledge the help and the guidance of my project supervisor Eric McCreath. Without his sunny disposition on a Monday afternoon this would have been a much more difficult endeavour!

Contents

Chapter 1 Introduction

The Comp3006 research project is the opportunity for a student to investigate any one aspect of computer science. The original plan for this project was to develop an application for the Android phone to act as a controller for the PlayStation 3 (PS3). However after analysis of the Android development platform, it became apparent that creating a simple Bluetooth application was trivial and the main difficulty lay in the reverse engineering of the PS3 Bluetooth communication. Therefore I chose to analyse the Bluetooth communication model and the feasibility of sniffing secure Bluetooth traffic.

1.1 The Problem

Bluetooth has been designed to be far more secure than any of the initial Wifi communication specifications. Wifi devices were easily changed to promiscuous mode[1] - the mode in which the network card passes all traffic to the CPU, rather than just the packets addressed to the device. As a result, Wifi and its securities were more easily cracked as deep packet analysis on all traffic was available.

Therefore the problem analysed in this report is to determine if Bluetooth can be easily sniffed from a consumer perspective; even when set up in security mode 3 - link-layer PIN authentification and MAC address encryption[2]. Bluetooth devices all lack a promiscuous mode and hence the ability to scan all traffic has made the simple interception of packets a lot harder; generally a Bluetooth radio can only receive packets which are directed at them. The second problem confronted in this report is the reverse engineering of the Bluetooth communication between the PS3 and its controllers and the feasibility of developing a software communicator.

1.2 Overview

Chapter 1 presents an outline of the report's focus, problem and contribution.

Chapter 2 contains the background of the project - providing an outline of Bluetooth and its various securities. It also provides a background on how this links to the Sony PlayStation 3.

In Chapter 3 the process of Bluetooth Sniffing is examined on an ideal test case - where both MAC addresses are known and the devices are in discoverable mode. The assorted tools needed to perform this task and break the encryption are investigated, such as the process of creating a Bluetooth sniffer and discovering the pairing process.

Chapter 4 utilises Spill and Bittau's work[3] on sniffing to determine an unknown MAC address of a device in undiscoverable mode. In doing so, this chapter analyses how the MAC address is encrypted into the packets of already paired devices.

In Chapter 5, a variety of ways (from Bluetooth Sniffing to analysis of the USB communication) are attempted to reverse-engineer the PS3 controller communication. This Chapter also illustrates the implementation of a PS3 input proxy - essentially a man-in-the-middle attack on the communication.

Chapter 6 provides the results of the analysis of the communication packets between controller and PS3.

Chapter 7 discusses the plausibility of furthering this research into a viable software controller and different avenues to explore in future projects.

1.3 Contribution

- Illustrate how consumer level Bluetooth devices can be transformed into Bluetooth sniffers.
- Demonstrate a basic Bluetooth attack on the pairing process of two Bluetooth phones.
- Use Spill and Bittau's[3] work to decipher the MAC address of a phone in undiscoverable mode.
- Utilise a combination of Bluetooth and USB sniffing to discover the Bluetooth MAC address of both the PS3 and PS3 controller.
- Set up an input proxy between the PS3 and its controller as a man-in-the-middle attack and passively record all communication packets.
- Provide an examination of these packets and the relationship between the bytes and button presses.

1.4 Limitations

There were several limitations in this project; most notably the lack of information regarding Bluetooth Sniffing and the lack of knowledge regarding the PS3 controller and its method of communication. Subsequently, this forced many hours of research into seemingly basic ideas (such as interfacing).

As the Sony PlayStation 3 is a closed system communicating only with official hardware, debugging was extremely difficult with no error messages - any problem in the communication could be numerous things ranging from an issue with the Bluetooth sniffer, to incorrect parameters to the PS3 rejecting all packets silently as they were incorrectly formed.

The only hardware limitation for this project was that of a dedicated USB sniffer. With an external USB sniffer, it would be possible to sniff the communication between controller and PS3 when connected via USB and perhaps it would be possible to reverse-engineer the USB control messages required to pair the device with the PS3.

Chapter 2

Theory

Bluetooth

Bluetooth is a specification for low-power radio communications for short distances, operating in the license-free ISM (industrial, scientific and medical) band at 2.4GHz[4]. As this is a very crowded communication frequency, the Bluetooth protocol uses a radio technology called Frequency-Hopping Spread Spectrum (FHSS)[5] to avoid interference. FHSS is a method of transmitting radio signals by rapidly switching the device among the 79 frequency channels using a pseudorandom sequence dependant on the MAC address of the master device. Bluetooth changes between the 1MHz wide channels up to 1600 times per second, making Bluetooth sniffing (the act of intercepting the data transmission) extremely difficult.

When a device connects to the Bluetooth piconet (a network consisting of all devices sharing the same Bluetooth FHSS hopping sequence and timing), the device must first synchronize to the network. The hopping sequence is established by the address of the master device and the phase of the hopping sequence determined by the internal clock of the master device. Hence to communicate in the Bluetooth network the device must synchronise to these values.

In a normal Bluetooth connection handshake, the slave receives one packet (the FHS synchronisation packet) containing the master's MAC address and clock. Packet exchange is based on the basic clock, which ticks at 312.5us intervals. A slot is defined as two ticks and in the case of single-slot packets (packets can be 1,3,5 slots long) the master transmits in even slots and receives in odd slots. The slave works conversely. At any single time the master communicates with one device at a time.[3]

2.1 Bluetooth Security

2.1.1 Data Whitening

All Bluetooth packets are 'whitened' (the act of scrambling the header and payload) before transmission. This is done in order to reduce highly redundant data, minimise DC bias and act as a security measure. The data is scrambled by a data whitening word and then unscrambled using the same word at the reciever[6]. In the case of Bluetooth the word used to whiten is the lower six bits of the clock, which is known only to the devices in the communication.

2.1.2 Bluetooth Pairing

Bluetooth pairing is used to control which devices can connect to a given Bluetooth device. This acts as a security mechanism, reducing the exposure of private data or unwanted control of the device. Hence two devices must be paired to communicate with each other.

The pairing process involves creating a shared secret known as the link key. Once this link key is created and known to both devices, the two devices are effectively paired. Once paired, the device can authenticate the other device's identity using this link key. Also once this link key is created, the devices can change to encrypted communication again using this link key.

Hence if the link key and master MAC address are determined by an attacker, it can masquerade as the other device (by using MAC address spoofing) and also eavesdrop on all communication between the two genuine devices.[7]

Bluetooth Profiles

A Bluetooth profile specifies the general behaviours and possible applications that Bluetooth devices may use in the course of their communication. There are currently over 20 different profiles. A device must be compatible with the subset of Bluetooth profiles necessary to use the desired services. Depending on the profile, the devices use the Bluetooth technology in many different ways. Each profile may contain the following:

- dependencies on other formats
- suggested user interface formats
- specific parts of the Bluetooth protocol stack.

The most important part of the specification is the definition of the protocol stack. The profile specifies options and parameters at each layer of the stack.

The profile of greatest significance for the PlayStation 3 is the Human Interface Device Profile (HID) which provides support for devices such as joysticks and controllers. As such, it is specially designed to provide low latency communications with low power requirements. It is also a reimplementation of the USB HID provide which enables operating systems to re-use existing code designed for these devices.

The PlayStation 3

The PlayStation 3 is a 7th generation console from the Sony Corporation. The wireless controller (named SIXAXIS) has two analog sticks, an accelerometer, 2 analog triggers, 8 pressure sensitive buttons, pressure sensitive d-pad and 3 digital buttons. It contains an internal 3.7V Li-ion battery, which provides up to 30 hours of gaming on a full charge. The controller connects via USB or Bluetooth to the PS3, however the Bluetooth lacks a discovery mode vital to allowing other Bluetooth devices to connect to it.

This leads to the most important piece of information - the controller must pair via USB first to communicate with the PS3. This pairing process writes the PS3's Bluetooth address to memory on the controller, and it also adds the controller to the PS3's allowed devices list. Without this pairing process the PS3 ignores all communication from the controller.

Chapter 3

Implementation

Bluetooth sniffing is the act of observing the traffic between two Bluetooth devices transparently. Unfortunately, it is difficult to do - without the address of the master device, interception of all of the information is basically not feasible once pairing is complete. This is due to frequency hopping and data whitening which is based off this master address. Therefore this address, along with the link key are required to be able to extract any meaningful data.

3.1 Setting up the Bluetooth Adapter

Moser's paper 'Busting the Bluetooth Myth - Getting RAW access'[8] explains a method for turning certain ordinary Bluetooth dongles into a sniffer by flashing it with the firmware of a commercial sniffer.

All Bluetooth dongles with a Cambridge Silicon Radio chipset and external flash memory can be used. To backup the firmware for safety, DFUtool is used:

```
dfutool -d hci0 archive firmware-backup.dfu
```

and flash the Frontline sniffer firmware:

```
dfutool -d hci0 upgrade airsnifferdev46bc2.dfu
```

To make sure the Frontline tool recognises the device, the product ID and vendor ID must be changed to 0x0002 and 0x0a12 respectively. This is done with the open source tool, bccmd:

```
// Find the current Product ID
./bccmd psget -s 0x0000 0x02bf
// Set the Product ID
./bccmd psset -s 0x0000 0x02bf 0x0002
// Display the Current Vendor ID
./bccmd psget -s 0x0000 0x02be
// Set the Vendor ID
./bccmd psset -s 0x0000 0x02be 0x0a12
```

Once the device is running in raw mode it can begin the task of sniffing all packets in the vicinity regardless of their device destination. The dongle is then tested with various Bluetooth devices to make sure it is working correctly.

The legalities of this approach are a grey area - the Frontline firmware was previously released freely as it was believed only to be of use to the official Frontline Comprobe sniffers[9]. However after Moser's discovery, the firmware is no longer allowed to be downloaded (and protection built in to 'brick' any non-official device). For the purposes of this research I use an old version of the firmware and only in a pure research behaviour.

3.2 Initial Sniffing - Mobile Phone

To demonstrate a very basic attack I decided to the initial sniffing problem was to discover the PIN of two known MAC address devices. This turned out to be highly successful. The typical attack scenario on a pair of Bluetooth devices is as follows.

Once the BD_ADDR of one device is determined (methods for this are explored in Chapter 4) the attacker then forges the MAC address to imitate one of the devices. The sniffer then asks to pair, indicating it has no link key / PIN if it was already paired. The master (depending on its implementation of the Bluetooth) will forget the old pairing data and request a new link key from the genuine slave. The Bluetooth sniffer now captures the key-pairing exchange taking place.

3.2.1 Key Generation

Once the pairing process is sniffed, the packets are exported and searched for the following data which determines the encryption link-key:

LMP_IN_RAND

LMP_COMB_KEY (Master)

LMP_COMB_KEY (Slave)

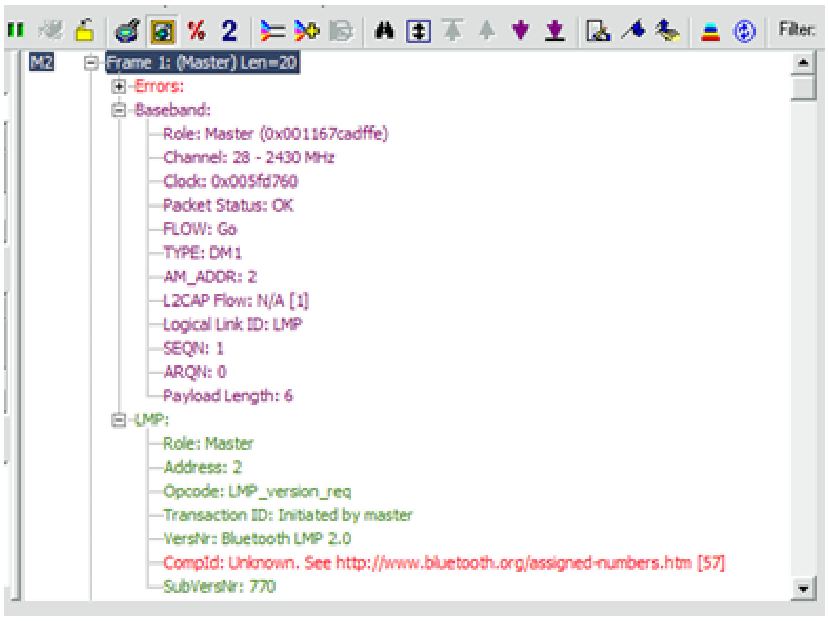
LMP_AU_RAND (Master)

LMP_AU_RAND (Slave)

LMP_SRES (Master)

LMP_SRES (Slave)

From these values (which are freely passed across the air) the link key can be recreated by brute force. Once the link key is discovered it is trivial to spoof our identity to the master or slave device.



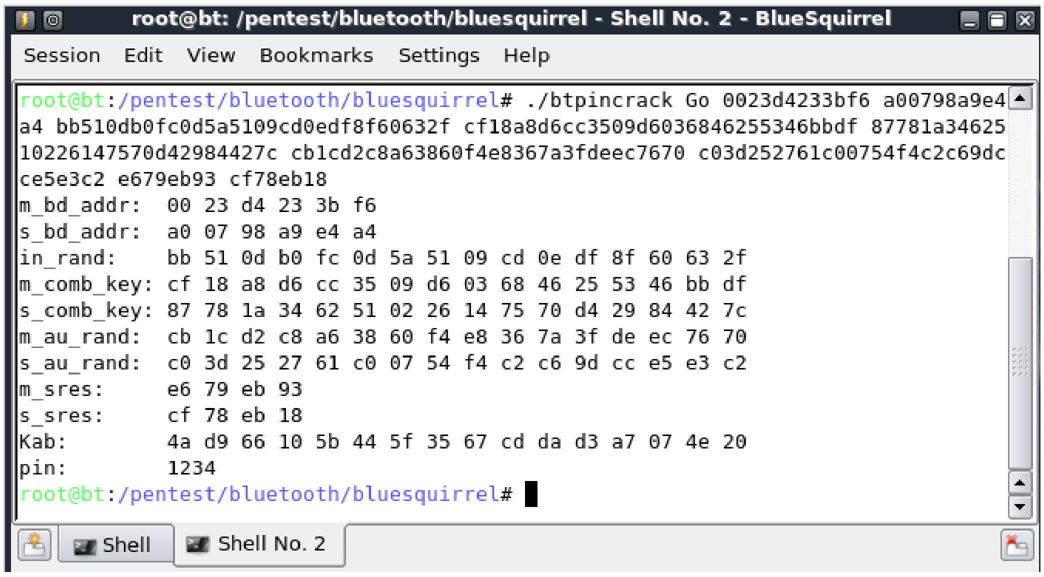
```
M2  Frame 1: (Master) Len=20
      Errors:
      Baseband:
        Role: Master (0x001167cadffe)
        Channel: 28 - 2430 MHz
        Clock: 0x005fd760
        Packet Status: OK
        FLOW: Go
        TYPE: DM1
        AM_ADDR: 2
        L2CAP Flow: N/A [1]
        Logical Link ID: LMP
        SEQN: 1
        ARQN: 0
        Payload Length: 6
      LMP:
        Role: Master
        Address: 2
        Opcode: LMP_version_req
        Transaction ID: Initiated by master
        VersNr: Bluetooth LMP 2.0
        CompId: Unknown. See http://www.bluetooth.org/assigned-numbers.htm [57]
        SubVersNr: 770
```

Figure 3.1: The Frontline Tool illustrating frame 1 of the S8003 phone

Example 1: Devices are HTC Magic Android and S8003 Samsung Diamond. These two devices were initially paired with an unknown PIN code. Firstly, their MAC address must be ascertained - which is trivially done by a device scan when the victim devices are in discovery mode (Figure 3.1).

The pairing process is initiated and the connection and pairing is sniffed as follows:

```
// Set the Device
csr_sniffer -d hci0
// Set the filter for the correct type of packets
csr_sniffer -d hci0 -f 7
// Set the Devices that are  to be sniffed
csr_sniffer -d hci0 -S 00:23:D4:23:3B:F6@A0:07:87:A9:E4:A4
// Begin sniffing, ignore zero packets and search for PIN data.
```


root@bt: /pentest/bluetooth/bluesquirrel - Shell No. 2 - BlueSquirrel
Session
Edit
View
Bookmarks
Settings
Help
Shell
Shell No. 2

```
root@bt:/pentest/bluetooth/bluesquirrel# ./btpincrack Go 0023d4233bf6 a00798a9e4
a4 bb510db0fc0d5a5109cd0edf8f60632f cf18a8d6cc3509d6036846255346bbdf 87781a34625
10226147570d42984427c cb1cd2c8a63860f4e8367a3fdeec7670 c03d252761c00754f4c2c69dc
ce5e3c2 e679eb93 cf78eb18
m_bd_addr:  00 23 d4 23 3b f6
s_bd_addr:  a0 07 98 a9 e4 a4
in_rand:    bb 51 0d b0 fc 0d 5a 51 09 cd 0e df 8f 60 63 2f
m_comb_key: cf 18 a8 d6 cc 35 09 d6 03 68 46 25 53 46 bb df
s_comb_key: 87 78 1a 34 62 51 02 26 14 75 70 d4 29 84 42 7c
m_au_rand:  cb 1c d2 c8 a6 38 60 f4 e8 36 7a 3f de ec 76 70
s_au_rand:  c0 3d 25 27 61 c0 07 54 f4 c2 c6 9d cc e5 e3 c2
m_sres:     e6 79 eb 93
s_sres:     cf 78 eb 18
Kab:        4a d9 66 10 5b 44 5f 35 67 cd da d3 a7 07 4e 20
pin:        1234
root@bt:/pentest/bluetooth/bluesquirrel# 
```

Figure 3.2: Using BTCrack to Bruteforce the PIN

```
csr_sniffer -e -z -p
```

All packets are now logged and Csr_sniffer will search for the particular series of packets containing the pairing procedure and extract the relevant data and output in a single line in the BTcrack format. BTcrack is an application developed by Shaked and Wool and improved by Thierry Zoller[10] that bruteforces the PIN/link key when supplied with the pairing data. BTcrack is then executed (see figure 3.2) on this data to bruteforce the PIN.

Once the encrypt key is discovered, all communication between the two devices can be logged. Masquerading as either device can now be accomplished using MAC address spoofing.

Chapter 4

Finding the MAC Address

Utilising the work of Spill and Bittau's "BlueSniff: Eve meets Alice and Bluetooth"[3], I show it is possible to sniff the connection between two devices when the MAC addresses are not known and they are not publicly broadcasting their Bluetooth status (i.e. not in discoverable mode).

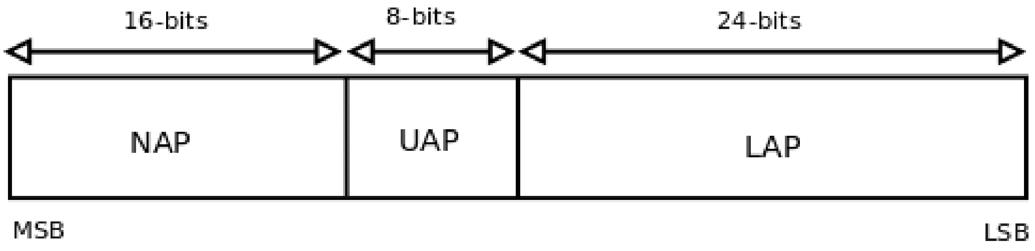


Figure 4.1: The form of a MAC address - Copyright Spill & Bittau

4.1 Data Whitening

As outlined in the 'whitening' section there are only 64 values which the scrambling can be based on. Spill's work illustrates a method to unwhiten the data with a simple function:

```
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;
 }
}
```

Code Copyright Spill and Bittau

This can be performed on any packet with a payload CRC. False positives are extremely unlikely (1 in 65536). Therefore for any packet, unwhiten is called and checked against the CRC. If it matches, the message is unscrambled and determining the MAC address can now begin.

The MAC address consists of three parts (as indicated in Fig. 4.1) and the entire MAC address is eight bytes: 2 bytes NAP, 1 byte UAP and 3 Bytes LAP.

4.1.1 LAP

The start of every packet is prefixed by an access code of 72 bits. The lower address part is found in the middle of every access code. Hence once unwhitened, the access code can be read directly from the packet and verified by the checksum.

4.1.2 UAP

The UAP generates the error check field (the HEC - header error code) in each packet's header. This HEC is calculated over the 10 bits of data in the header. The UAP from the master device initialises a register of 8 bits. Each bit of the header is fed into the register in the order of transmission, and this final calculation is appended to the end of the header.

The process to convert the UAP to HEC is a XOR operation and hence can be reversed.

```
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;
}
```

4.1.3 NAP

The NAP is only two bytes long and the first byte is generally zero. The 256 values can be brute forced by sending a Bluetooth connection to all possible addresses. However this can be simplified by noting that the NAP and UAP are based upon the vendor of the device and can be estimated by the potential sniffer.

4.1.4 Final Algorithm

1. Determine LAP from access code.
2. Generate 64 candidates by using the unwhitening algorithm for all possible 64 inputs.
3. Generate the 64 candidates for the UAP from the HEC from each packet.
4. For each of these pairs, calculate and compare the CRCs and find which candidate is correct.
5. Bruteforce the remaining NAP after first eliminating candidates with the vendor ID lookup.

Chapter 5

Reverse Engineering the PS3 Controller Protocol

5.1 Sniffing the PS3 Connection

The first major problem regarding the PS3 is that the pairing between the controller and the host PS3 is carried out through a direct USB connection. As the master MAC addresses are exchanged here and written to the controller which is registered to the PS3, this provides some security by obfuscation. I attempted to investigate whether it was possible, and perhaps easier, to sniff the USB connection (rather than the Bluetooth) connection to determine the MAC addresses.

5.1.1 USB Sniffing

USB sniffing is remarkably simple compared to Bluetooth sniffing because it is done by open source software rather than hardware. USB sniffers work by creating an intermediary service between the device and the operating system, silently observing all control and interrupting and bulking USB requests. The program used was USB Snoop. While this revealed some of the control points (see (Figure 5.1) it did not reveal the correct control points for the Bluetooth master address.

5.1.2 PS3 USB Connection

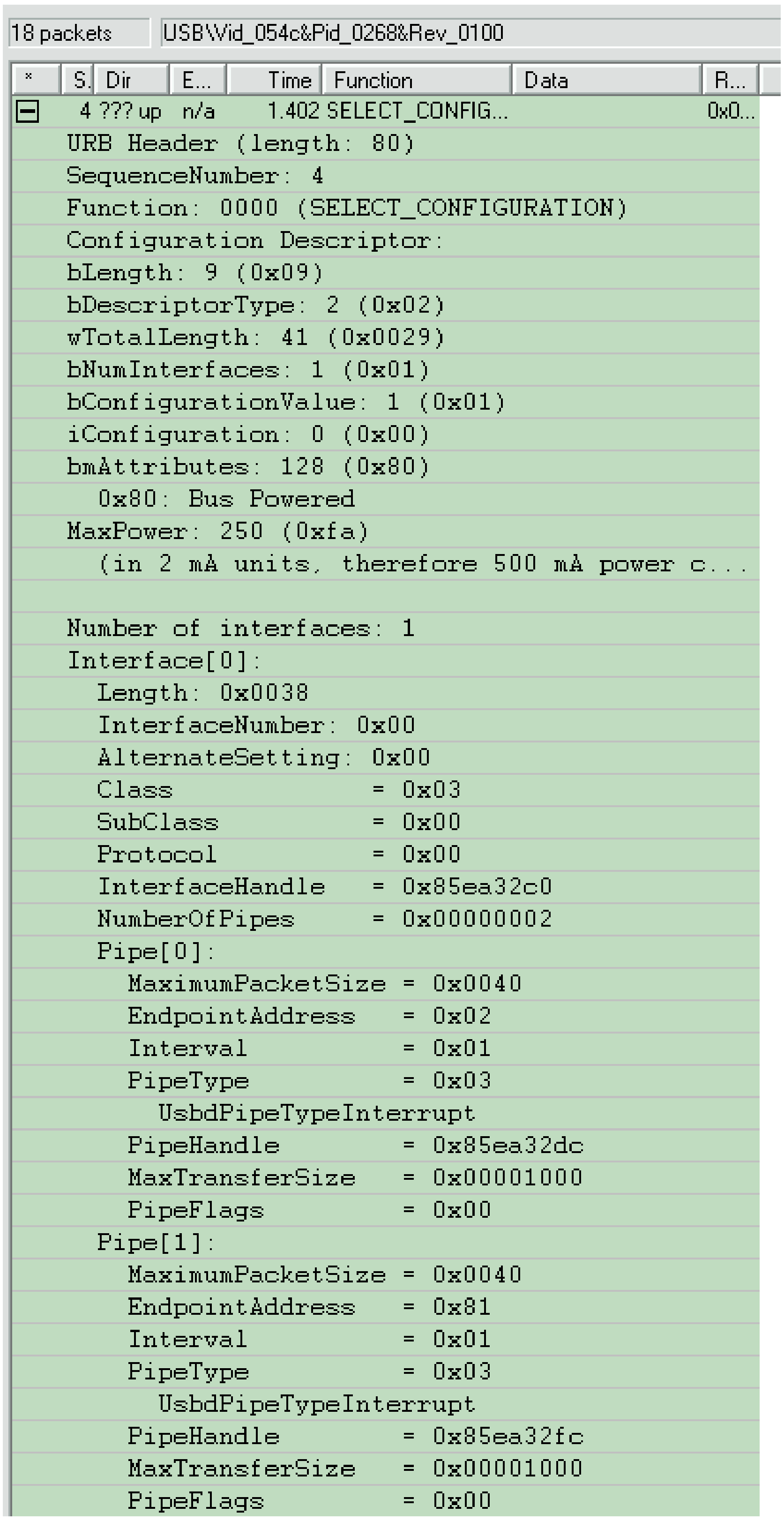
These USB control requests show that there exists two separate interrupt pipes to interact with the device - 0x02 and 0x81. However, the control requests must be determined to command the controller to parse data to these interrupt pipes. As the controller connects to the PC as a Human Interface Device (HID) it must respond to HID commands. With this knowledge I researched the Bluetooth USB control requests and found the work of Pascal[11] and his sixpair program. This contained two routines - one which extracted the current Bluetooth master address (using HID_GET_REPORT as the bmRequest) and one to write a new address to the controller's memory (using HID_SET_REPORT).

```
char bdaddr[8]= { 0x01, 0x00, bdaddr[0],bdaddr[1],
bdaddr[2],bdaddr[3],bdaddr[4],bdaddr[5] };
int res = usb_control_msg (devh, USB_DIR_OUT | USB_TYPE_CLASS |
USB_RECIP_INTERFACE, HID_SET_REPORT, 0x0300 | PS3_F5_REPORT_ID,
itfnum, msg, sizeof(msg),5000);
```

The important thing to note here:

```
USB_DIR_IN | USB_RECIP_INTERFACE = 0x81
```

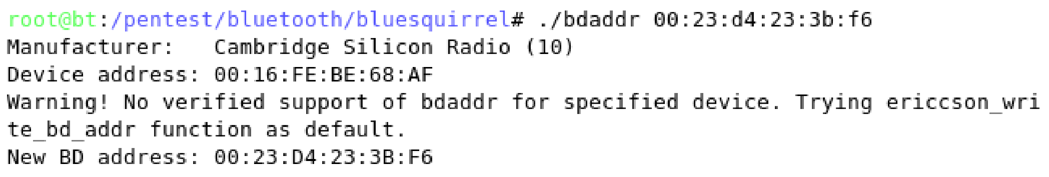
The last address that must be retrieved is the controller's address. This is discovered by changing the master address that the controller attempts to communicate with to the address of our Bluetooth dongle. Hcidump will log



```
18 packets   USB\Vid_054c&Pid_0268&Rev_0100
*   S. Dir     E...   Time Function           Data                 R...
[-]  4 ??? up  n/a    1.402 SELECT_CONFIG...                        0x0...
    URB Header (length: 80)
    SequenceNumber: 4
    Function: 0000 (SELECT_CONFIGURATION)
    Configuration Descriptor:
    bLength: 9 (0x09)
    bDescriptorType: 2 (0x02)
    wTotalLength: 41 (0x0029)
    bNumInterfaces: 1 (0x01)
    bConfigurationValue: 1 (0x01)
    iConfiguration: 0 (0x00)
    bmAttributes: 128 (0x80)
      0x80: Bus Powered
    MaxPower: 250 (0xfa)
      (in 2 mA units, therefore 500 mA power c...

    Number of interfaces: 1
    Interface[0]:
      Length: 0x0038
      InterfaceNumber: 0x00
      AlternateSetting: 0x00
      Class              = 0x03
      SubClass           = 0x00
      Protocol           = 0x00
      InterfaceHandle    = 0x85ea32c0
      NumberOfPipes      = 0x00000002
      Pipe[0]:
        MaximumPacketSize = 0x0040
        EndpointAddress   = 0x02
        Interval          = 0x01
        PipeType          = 0x03
          UsbdPipeTypeInterrupt
        PipeHandle        = 0x85ea32dc
        MaxTransferSize   = 0x00001000
        PipeFlags         = 0x00
      Pipe[1]:
        MaximumPacketSize = 0x0040
        EndpointAddress   = 0x81
        Interval          = 0x01
        PipeType          = 0x03
          UsbdPipeTypeInterrupt
        PipeHandle        = 0x85ea32fc
        MaxTransferSize   = 0x00001000
        PipeFlags         = 0x00
```

Figure 5.1: The endpoints of the PS3 Controller



```
root@bt:/pentest/bluetooth/bluesquirrel# ./bdaddr 00:23:d4:23:3b:f6
Manufacturer:   Cambridge Silicon Radio (10)
Device address: 00:16:FE:BE:68:AF
Warning! No verified support of bdaddr for specified device. Trying ericcson_wri
te_bd_addr function as default.
New BD address: 00:23:D4:23:3B:F6
```

Figure 5.2: Using Bdaddr to forge the address

all incoming connection requests, which will include the unique MAC address of the controller.

5.1.3 PS3 Bluetooth Sniffing

The first attempt to connect to the PS3 directly from the PC resulted in failure. The PS3 restricts access to paired devices and also to a specific class of devices, the class 0x508 (game controller). Therefore using the Bdaddr program (see Figure 5.2) the MAC address of the dongle is changed to the controller's MAC address:

This attempt results in moderate success - the PlayStation accepts the connection request but quickly shuts down all communication as soon as the correct setup packets are not communicated.

Directly sniffing the communication between the two devices was also unsuccessful as the devices are already paired via USB and do not respond to Bluetooth Page requests for a new pairing. Without discovering the link-key following the Bluetooth communication was impossible. The only option to discover this would be to USB sniff the actual pairing process.

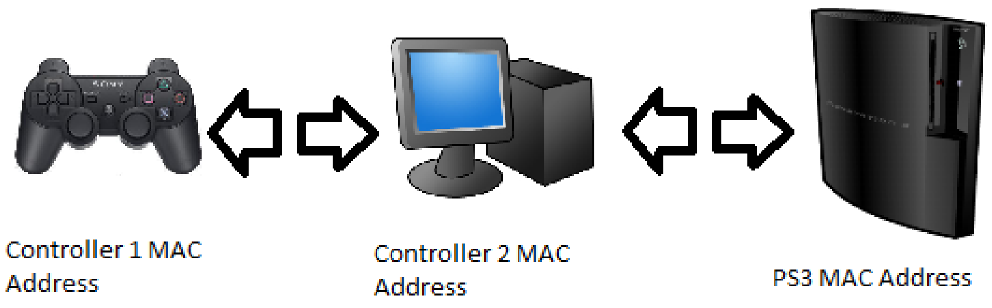


Figure 5.3: Illustrating the Proxy Setup

5.1.4 PS3 Proxy

The solution to these problems came when I discovered sixproxy - a program based upon sixpair written by Mikael Bouillot[12]. The concept of this program is to get the PS3 to act as a proxy between the PS3 and the controller. Originally designed to remove the deadzone in the PlayStation 3 accelerometer I saw that it would be able to resolve the communication rejection problem. This works by finding the MAC address of controller 2, and MAC address spoofing the PC's Bluetooth address to controller 2. Aftwerwards PS3 address in controller 1 is set to that of controller 2. Hence the controller thinks it's connecting to the PS3, and the PS3 thinks its connecting to controller 2, when in fact they are both connecting to the PC.

I decided to implement this idea myself but as a man-in-middle attack - transparently recording the packets between the controller and the PS3. This proved highly successful. The PlayStation 3 believed it was communicating with the controller directly and hence eliminated the disconnect issues. As a result it was possible to capture all the packets when specific buttons were pressed down. Unfortunately, at first it appeared that the packets were encrypted / obfuscated as there appeared no relationship between button press and packet data. Thankfully with deeper analysis, patterns began to emerge between button press and byte data.

Chapter 6

Results

6.1 Deconstructing the Packet Data

The process of retrieving meaningful data from the sniffed data was difficult. With no references or information available on the PS3 controller specification it was a process involving creating all possible button scenarios and finding patterns in the resulting data.

Table 6.1: Digital Buttons

Byte Number	Button(s) Controlled	Notes
	Mutiple buttons	
	Select - 1 = 2^0	
	Start - 2 = 2^3	Hence holding down
Byte[7]	Up - 16 = 2^4	left and up
	Right - 32 = 2^5	becomes 10010000.
	Down - 64 = 2^6	
	Left - 128 = 2^7	
	Mutiple buttons	
	L2 - 1 = 2^0	
	R2 - 2 = 2^1	
	L1 - 4 = 2^2	
Byte[8]	R1 - 8 = 2^3	
	Triangle - 16 = 2^4	
	Circle - 32 = 2^5	
	Cross - 64 = 2^6	
	Square - 128 = 2^7	
Byte[9]	PS Button - 1 = 2^0	

Table 6.2: Analog Sticks

Byte Number	Button(s) Controlled	Notes
Byte[10]	Horizonal Left Analog Stick	0 = Stick Held Left 128 = Stick in Centre 255 = Stick Held Right
Byte[11]	Vertical Left Analog Stick	0 = Stick Held Up 128 = Stick in Centre 255 = Stick Held Down

Table 6.3: Pressure Sensors

Byte Number	Button(s) Controlled	Notes
Byte[19]	Up	
Byte[20]	Right	
Byte[21]	Down	
Byte[22]	Left	
Byte[23]	L2	
Byte[24]	R2	
Byte[25]	L1	
Byte[26]	R1	
Byte[27]	Triangle	
Byte[28]	Circle	
Byte[29]	Cross	
Byte[30]	Circle	

Bytes 19-30 contain the pressure sensor reading, which indicates the amount of pressure downwards on the controller by the user on a particular button.

Table 6.4: Accelerometer

Byte Number	Button(s) Controlled	Notes
Byte[46] + Byte[47]	Pitch Accelerometer	A two byte value (0-65535) holding the controllers current horizontal inclination.
Byte[48] + Byte[49]	Roll Accelerometer	A two byte value (0-65535) holding the controllers current vertical inclination.
Byte[49] + Byte[50]	Yaw Accelerometer	A two byte value (0-65535) holding the controllers current vertical inclination.

Chapter 7

Discussion

While it was possible to intercept and reverse engineer some of the data of the packets, there still exists many limitations before a software emulated controller can be developed.

The main issues to overcome is the pairing with the PS3 and then the forging the initial connection parameters once a link is established. When a controller is initially connected to the PS3 and the PS button is pressed, the controller learns the Bluetooth MAC address of the PS3. Concurrently the PS3 learns the controllers address and adds this information to its allowed devices. This pairing process would have to be USB sniffed in some form, or through trial and error to find the correct parameters to pass to the PS3. However while this is an inconvenience from a user perspective, it can be mitigated during development with address spoofing.

The initial communication parameters passed in the Bluetooth connection however are crucial to the communication. In the first 30 packets (15 in each direction), information on clock, power status, encryption and status are exchanged. Although the PS3 will accept the a Bluetooth connection request from its list of known controller MAC address, it stops responding without the correct information.

Once these problems are rectified, it is entirely plausible to create a software controller. The button presses and other data are easily duplicated by writing to byte arrays, and this could be mapped to the keyboard.

Another option of potential research is using the fact that the PlayStation 3 controller is a HID device. Any other comparable HID device (i.e. a keyboard) should be able to be masqueraded as the controller. I believe this would be substantial effort to do correctly.

Throughout this project the use of the Bluetooth sniffer was key to understanding and analysing these packets. This was not without its own caveats - the sniffer was incredibly unreliable. It would reset, lock itself or otherwise become unresponsive, programs would have trouble interfacing with it, and it would ignore communications haphazardly. MAC address spoofing would also cause unreliability when in use and also randomly revert to its old address halfway through communication.

This research of sniffing the communications between the mobile phones and sniffing the pin relies on the devices using $\leq$ 2.0 Bluetooth. Bluetooth 2.1 introduces a new pairing mechanism labeled Secure Simple Pairing (SSP) and also requires encryption for all non-SDP (Service Discovery Protocol) connections. Instead of a simple PIN, SSP uses a form of public key cryptography to perform the pairing procedure. This is more difficult to determine the link-key but according to research should be possible. [13]

Chapter 8

Conclusion

I have shown through an analysis of open source tools that the Bluetooth communication can be easily sniffed with cheap consumer grade products, even when set in undiscoverable mode in an ideal situation. However applying these techniques in a real-world environment would be substantially more difficult as forcing the re-pair process is unreliable at best.

Using these techniques, I apply these penetrations tests to the Sony PlayStation 3 controllers. Unfortunately due to a combination of issues such as hardware failure and a different communication model for the HID controller device, I was unsuccessful in the raw sniffing of the MAC address. I overcame these problems by utilising USB sniffing (finding the controller control commands) and then following with MAC address spoofing and a Bluetooth input proxy to sniff the communication packets.

Logging all packets through the PC I was able to discern the controller specification and provide mappings between the byte arrays and button presses. The initial packets (containing the setup and authentication data) were unable to be reverse-engineered, however with further research a software emulated controller is a definite possibility.

Bibliography

[1] TamoSoft. *Promiscuous Monitoring in Ethernet and Wi-Fi Networks*, 2005. `http://www.ferret.com.au/n/Secure-wireless-networks-for-industrial-applications-n690552`.

[2] Konstantin Sapronov. Bluetooth, bluetooth security and new year war-nibbling, May 2007. `http://packetstormsecurity.org/papers/wireless/busting_bluetooth_myth.pdf`.

[3] Dominic Spill and Andrea Bittau. Bluesniff: Eve meets alice and bluetooth. In *WOOT '07: Proceedings of the first USENIX workshop on Offensive Technologies*, pages 1–10, Berkeley, CA, USA, 2007. USENIX Association.

[4] Wikipedia. Bluetooth — wikipedia, the free encyclopedia, 2010. `http://en.wikipedia.org/w/index.php?title=Bluetooth`.

[5] Bruno Forgue. Secure wireless networks for industrial applications, May 2004. `http://www.ferret.com.au/n/Secure-wireless-networks-for-industrial-applications-n690552`.

[6] David Blankenbeckler. An introduction to bluetooth, 2010. `http://www.wirelessdevnet.com/channels/bluetooth/features/bluetooth.html`.

[7] Christian Gehrmann. Bluetooth security white paper, 2002. `http://www.bluetooth.com/Research%20and%20White%20Papers/security_whitepaper_v1.pdf`.

[8] Max Moser. Busting the bluetooth myth - getting raw access, May 2006. `http://www.ferret.com.au/n/Secure-wireless-networks-for-industrial-applications-n690552`.

[9] Frontline. Fts4bt bluetooth protocol analyzer and packet sniffer, 2010. `http://www.fte.com/products/fts4bt.aspx`.

[10] Yaniv Shaked and Avishai Wool. Cracking the bluetooth pin. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 39–50, New York, NY, USA, 2005. ACM.

[11] Pascal. Using the playstation 3 controller in bluetooth mode with linux, 2010. `http://www.pabr.org/sixlinux/sixlinux.en.html`.

[12] Mikael Bouillot. Ps3 bluetooth input proxy, August 2009. `http://www.corbac.com/page43.html`.

[13] Andrew Y. Lindell. Attacks on the pairing protocol of bluetooth v2.1, 2008.