HOW TO USE OLD GSM PROTOCOLS/ENCODINGS TO KNOW IF A USER IS ONLINE ON THE GSM NETWORK AKA PINGSMS 2.0

2015-07-27 | #SmsManager, #SmsManager.sendDataMessage, #delivery report, #gsm, #mms, #pdu, #ping sms, #sendDataMessage, #sms, #wap, #wap push, #wap push notifications

In the last few months I've been playing with Android's low level GSM API, a few years ago the (in)famous **sendRawPdu** API was available, allowing a developer to manually encode a SMS message at a very low level before sending it to the GSM baseband itself and quite a few applications sending all kind of weird SMS (flash sms, silent sms, etc) were born (for a brief overview of PDU encoding refer to this page).

(Un)fortunately Google decided to remove that API, it's still not sure if they did it for security related purposes or during some refactoring of their IPC IBinder mechanism, but nowadays it's no more available unless you use some very old phones/firmwares (on most devices they removed the ttyUSB serial interfaces to send AT commands to the GSM modem as well).

Until a couple of months ago, when I found the **SmsManager.sendDataMessage** API which, apparently, it's not used anywhere (if you search for it you'll find only a few examples, but nothing regarding how to use it with manually encoded PDUs).
Using this API we're able to manually encode our SMS, moreover we can specific a "port" as one of its arguments which will identify what kind of sms we're gonna send, in this post I'll talk about port 2948, namely the port used to send **WAP PUSH notifications**.

WAP PUSH messages were an old mechanism to basically force a remote device to visit a URL encoded in the SMS payload itself (I know, security wise it was very dumb, but we're talking about the 90s), this specific request is called a "**Service Load**" (SL) request, where an XML payload like the following was encoded in a binary form and sent along the PDU to the device.

```
<?xml version="1.0"?>
<!DOCTYPE sl PUBLIC "-//WAPFORUM//DTD SL 1.0//EN"
                    "www.wapforum.org/DTD/sl.dtd">
<sl href="http://some-evil-site.com/evil-content.html"/>
```

Nowadays this protocol is no more handled for obvious reasons … I said **handled** and not **supported** because the GSM basebands software is actually still able to receive it, but the higher level software (the OS and its components) will simply ignore it.
For instance, on Android 5.0 you can see the following logs (`logcat -b radio`) when the device receives such payload:

```
D/WAP PUSH( 1287): Rx: 0a0603...
D/RILC    (  185): SOCKET RIL_SOCKET_1 REQUEST: SMS_ACKNOWLEDGE length:20
D/RILC    (  185): RequestComplete, RIL_SOCKET_1
E/RILC    (  185): Send Response to RIL_SOCKET_1
D/RILJ    ( 1287): [9277]< SMS_ACKNOWLEDGE  [SUB0]
V/WAP PUSH( 1287): appid found: 2:application/vnd.wap.slc
W/WAP PUSH( 1287): wap push manager not found!
V/WAP PUSH( 1287): fall back to existing handler
V/WAP PUSH( 1287): Delivering MMS to: com.google.android.talk
com.google.android.apps.hangouts.sms.MmsWapPushDeliverReceiver
```

So the event is delivered to the Google Hangouts application (the default SMS/MMS handler on my phone) which simply **will ignore this kind of payloads** unless they are simple MMS instead of anything else (WAP PUSHes in our case):

```
public class MmsWapPushDeliverReceiver extends BroadcastReceiver
{
  public void onReceive(Context paramContext, Intent paramIntent)
  {
    if (("android.provider.Telephony.WAP_PUSH_DELIVER".equals(paramIntent.getAction())) &&
("application/vnd.wap.mms-message".equals(paramIntent.getType())))
      RealTimeChatService.a(paramIntent.getByteArrayExtra("data"));
  }
}
```

This means that, although the device will receive the data, **no kind of notification will be shown to the user** and the data itself won't be saved anywhere in the system, but simply discarded.

Having said that, there's a tiny detail that's very handy for us … after delivering the WAP PUSH message, the destination **operator BTS will reply to us with a delivery report**. This report will be sent only if the device is turned on and completely able to receive the message (turned on and with enough GSM network coverage).
In fact, the sendDataMessage API accepts as its last argument a delivery PendingIntent, in other words our application will be informed as soon as the delivery report will be sent back.

We can take advantage of this to do the following:

- Craft a WAP PUSH message encoding it manually.
- Pass it to the sendDataMessage API and register a delivery intent.
- The WAP PUSH will be sent to the target mobile phone and we'll receive the delivery notification if the phone is turned on, if it's not we'll receive it as soon as it will be turned on.
- The target user won't notice absolutely anything.

So, we can basically track a target user GSM network activity invisibly, knowing exactly when the target's device is turned on without him having a single chance to notice anything.

Here's a very simple PoC application I've made to show how to use such API, the same kind of PDU can be sent using a normal GSM serial dongle and some software like Gnokii.