This blog is now hosted on a GPS/LTE modem

Apr 1, 2021

No, really. Despite the timing of this article, this is not an April Fool's joke.

PinePhone's GPS/WWAN/LTE modem

While developing software on the PinePhone, I came across this peculiar message in `dmesg`:

```
[   25.476857] modem-power serial1-0: ADB KEY is '41618099' (
```

For context, the PinePhone has a Quectel EG25-G modem, which handles GPS and wireless connectivity for the PinePhone. This piece of hardware is one of the few components on the phone which is closed-source.

When I saw that message and the mention of ADB, I immediately thought of Android Debug Bridge, the software commonly used to communicate with Android devices. "Surely," I thought, "it can't be talking about *that* ADB". Well, turns out it is.

The message links to an article which details the modem in question. It also links to an unlocker utility which, when used, prints out AT commands to enable `adbd` on the modem.

```
$ ./qadbkey-unlock 41618099
AT+QADBKEY="WUkkFzFSXLsuRM8t"
AT+QCFG="usbcfg",0x2C7C,0x125,1,1,1,1,1,1,0
```

These can be sent to the modem using `screen`:

```
# screen /dev/ttyUSB2 115200
```

For whatever reason, my input wasn't being echoed back, but the screen session printed out "OK" twice, indicating it had executed the commands fine.

After setting up proper `udev` rules and `adb` on my "host machine", which is the PinePhone, the modem popped up in the output for `adb devices`, and I could drop into a shell:

```
$ adb devices
List of devices attached
(no serial number)      device

$ adb shell
/ #
```

Because `adbd` was running in root mode, I dropped into a root shell. Neat.

It turns out the modem runs its own OS totally separate from the rest of the PinePhone OS. With the latest updates, it runs Linux 3.18.44.

Running a webserver

For whatever reason, I thought it'd be fun to run my blog on this thing. Since we were working with limited resources (around 48M of space and the same amount of memory), and the fact that my blog is just a bunch of static files, I decided that something like nginx (as lightweight as it is) would be a bit overkill for my purposes.

darkhttpd seemed to fit the bill well. Single binary, no external dependencies, does GET and HEAD requests only. Perfect.

I used the armv7l-linux-musleabihf-cross toolchain to cross compile it for ARMv7 and statically link it against musl. `adb push` let me easily push the binary and my site assets to the modem's `/usrdata` directory, which seems to have a writable partition about 50M big mounted on it.

The HTTP server works great. I decided to use ADB to expose the HTTP port to my PinePhone:

```
$ adb forward tcp:8080 tcp:80
```

As ADB-forwarded ports are only bound to the loopback interface, I also manually exposed it to external connections:

```
# sysctl -w net.ipv4.conf.all.route_localnet=1
# iptables -t nat -I PREROUTING -p tcp --dport 8080 -j DNAT -
```

I could now access my blog on `http://pine:8080/`. Cool!

Throughput?

I ran `iperf` over ADB port forwarding just to see what kind of throughput I get.

```
$ iperf -c localhost
------------------------------------------------------------
Client connecting to localhost, TCP port 5001
```

```
TCP window size: 2.50 MByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 44230 connected with 127.0.0.1 por
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.6 sec  14.4 MBytes  11.4 Mbits/sec
```

So around 10Mb/s. Not great, not terrible.

The PinePhone itself is connected to the network over USB (side note: I had to remove two components from the board to get USB networking to work). Out of interest, I ran `iperf` over that connection as well:

```
$ iperf -c 10.15.19.82
------------------------------------------------------------
Client connecting to 10.15.19.82, TCP port 5001
TCP window size:  136 KByte (default)
------------------------------------------------------------
[  3] local 10.15.19.100 port 58672 connected with 10.15.19.8
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.4 sec  25.8 MBytes  20.7 Mbits/sec
```

Although I was expecting more, it doesn't really matter, as I was bottlenecking at the ADB-forwarded connection.

Further thoughts

I wonder how secure the modem is. It turns out a lot of AT commands use `system()` on the modem. I suspect some of those AT commands may be vulnerable to command injection, but I haven't looked into this further. It also doesn't really matter when dropping into a root shell using ADB is this easy.

At first glance, this seems like a perfect method to obtain persistence for malware. With root access on the host system, malware could implant itself into the modem, which would enable it to survive reinstalls of the host OS, and snoop on communications or track the device's location. Some of the impact is alleviated by the fact that all interaction with the host OS happens over USB and I2S and only if the host OS initiates it, so malware in the modem couldn't directly interact with the host OS.

Author | Rasmus Moorats

Ethical Hacking and Cybersecurity professional with a special interest for hardware hacking, embedded devices, and Linux.