ERNW
providing security.

ERNW WHITE PAPER 56/ OCTOBER, 2016

CISCO AIRONET 602 OFFICEEXTEND ACCESS POINT – MIC AND PRIVATE KEY EXTRACTION VULNERABILITY

Version:	1.0
Date:	10/04/2016
Author(s):	Stefan Kiese

Matthias Luft
Digitally signed by Matthias Luft
Date: 2016.10.05 10:30:31 +02'00'

ERNW Enno Rey Netzwerke GmbH
Carl-Bosch-Str. 4
69115 Heidelberg

www.ernw.de
www.troopers.de
www.insinuator.net

ERNW
providing security.

TABLE OF CONTENT

1 INTRODUCTION 4

2 ENVIRONMENT 5

2.1 Lab 5

2.2 Hardware 5

2.3 Software 5

3 INFORMATION GATHERING 6

3.1 Vendor Information 6

3.2 Disassembling the Hardware 7

4 GAINING ACCESS 9

4.1 Serial Console, Part I 9

4.1.1 Proof of Concept 10

4.2 Serial Console, Part II 11

4.3 NAND Glitching 12

4.3.1 Proof of Concept 13

4.4 CFE (Common Firmware Environment) 14

4.5 BusyBox 16

4.6 Extracting the MIC Certificate from the Access Point 16

4.7 JTAG 18

5 CONCLUSION & OUTLOOK 20

6 APPENDIX 21

6.1 Disclosure Timeline 21

6.2 Device Information 22

6.3 Disclaimer 26

ERNW
providing security.

LIST OF FIGURES

Figure 1: Cisco Aironet 602 PCB ____ 7
Figure 2: JTAG/UART Pin Headers ____ 8
Figure 3: Bridged JTAG Connector ____ 9
Figure 4: Connected Serial Cable ____ 10
Figure 5: Grounding Wire and Isolated Pin at Flash Memory ____ 12
Figure 6: NAND Flash Pinout ____ 13

1 Introduction

In this white paper we will cover not only the results of a hardware penetration test which has started as a customer project and ended up in a research project, but also the steps leading to them.

The subject of this test was the Cisco Aironet 602 OfficeExtend Access Point[1] (short: Cisco OEAP602), which is intended to extend the corporate network to e.g. teleworker's home via DTLS.

Initially we were asked by our customer to have a look on the device and to find out if there is any chance to dump the certificates and keys stored on the device. It turned out that there are several possibilities to do this as you can read on the following pages.

What we found is a so called "Manufacturer Installed Certificate", or short "MIC" and the corresponding private key. Both could be dumped via UART and JTAG access from the AP's flash memory device. There is also a third method to dump the NAND flash using third party tools, but this costs more time due to electrical interferences when dumping in-circuit.

There might also be some additional vulnerabilities (e.g. use of old and vulnerable software versions) which were not in our scope. We only focused on the hardware hacking part. The vulnerabilities we have found went through a responsible disclosure process with Cisco. They (Cisco) made the case public on 22nd of September 2016.

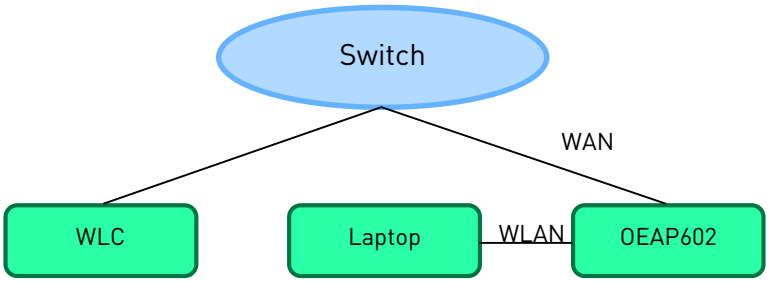
[1] *http://www.cisco.com/c/en/us/products/wireless/aironet-600-series-officeextend-access-point/index.html*

2 Environment

First, in this chapter we give a brief description about the used environment while doing the penetration test.

2.1 Lab

The following lab environment was used to conduct the security assessment. It resembles a typical customer environment.



2.2 Hardware

For our test setup, the hardware listed below has been used:

- Cisco Office Extend Access Point:
  - Cisco AIR-OEAP602I-E-K9
- Cisco Wireless Controller:
  - Cisco AIR-CT2504-K9 V01

2.3 Software

The following software versions were installed on the access point and used on the WLC:

- WLC:
  - WLC 8.0.110.0
- AP:
  - CFE 5.10.144.16 Evora
  - Kernel 2.4.20
  - Evora OEAP version 7.0.112.71 (compiled Feb 27 2011 at 23:11:19)
  - BusyBox v0.60.0 (2011.02.28-07:10+0000)

3 Information Gathering

As the first step, we started with some information gathering. This includes information provided by the manufacturer such as product datasheets or similar public available information. Another helpful source for information is the FCC[2], while looking for wireless devices.

3.1 Vendor Information

The target of evaluation was a Cisco AIR-OEAP602I-E-K9 and the first step was to gather publicly available information provided by Cisco to evaluate whether the datasheet[3] might be useful to get more detailed information about the hardware. Unfortunately, the datasheet did not provide much usable information (besides the radio interfaces) about the built-in hardware. The only information that was given, was the disabled USB port (which is hidden behind a sticker) and that the AP got 64MB DRAM and 16MB of flash memory.

[2] *Federal Communications Commission, https://www.fcc.gov/about-fcc/what-we-do*
[3] *http://www.cisco.com/c/en/us/products/collateral/wireless/aironet-600-series-officeextend-access-point/data_sheet_c78-651456.pdf*

3.2 Disassembling the Hardware

As the datasheet didn't provide any useful information, the access point was disassembled to have a closer look at the PCB (Printed Circuit Board). Fortunately, this was easily possible as the bottom and top part were only clipped.

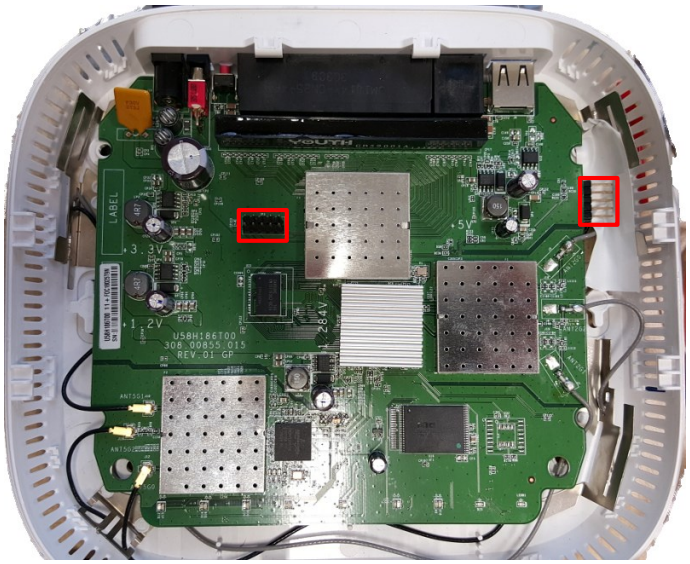


Figure 1: Cisco Aironet 602 PCB

After having a closer look at the PCB's top and bottom sides and removing some RF-shieldings, we encountered some Broadcom chips (datasheets are rare concerning this vendor), RAM and a TSOP NAND flash memory[4]. Of much higher interest were some unused 0.1" pads and holes in typical pin header assembly (5x1 and 5x2), which we assumed to be UART[5] (serial port, like the one used by PCs but with different logic levels) and JTAG[6] (port for debugging, flashing firmware and testing).

To figure out whether these ports could be useful, we were doing some voltage measurements while the access point was running and got some pins where 3.3V could be measured. We soldered some pin headers on these points for the next steps as shown in the figure below.

[4] *Macronix MX29GL128ELT2I-9DG, http://media.digikey.com/pdf/Data%20Sheets/Macronix/MX29GL128E.pdf*
[5] *https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter*
[6] *https://en.wikipedia.org/wiki/Joint_Test_Action_Group*

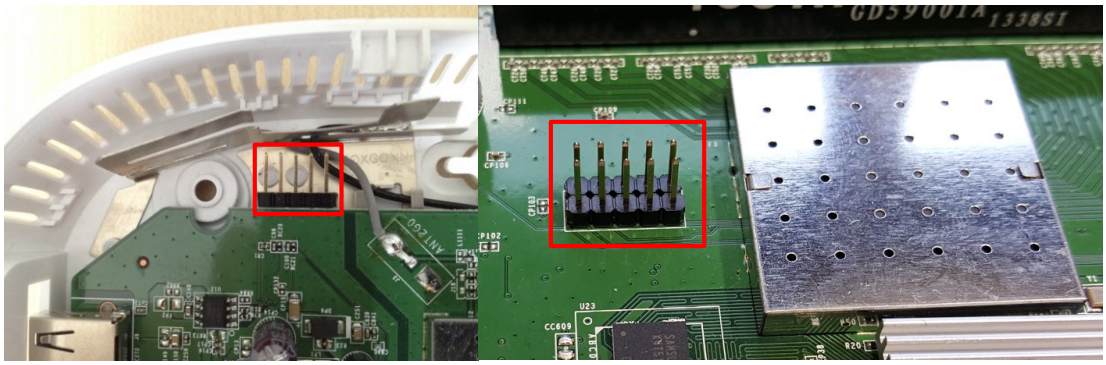


Figure 2: JTAG/UART Pin Headers

4 Gaining Access

Using the JTAGulator[7], which is an easy to handle hardware tool used for brute forcing of possible JTAG and UART connectors, it is very easy to identify JTAG and UART pinout. Doing so, it was possible to identify UART on the bottom side 5x1-connector, whereas the expected JTAG connector seemed to be dead. By manually following the traces from the JTAG connector to the IC it became apparent that the traces were interrupted, so we used bridged soldering pads to make the JTAG Connecter usable.

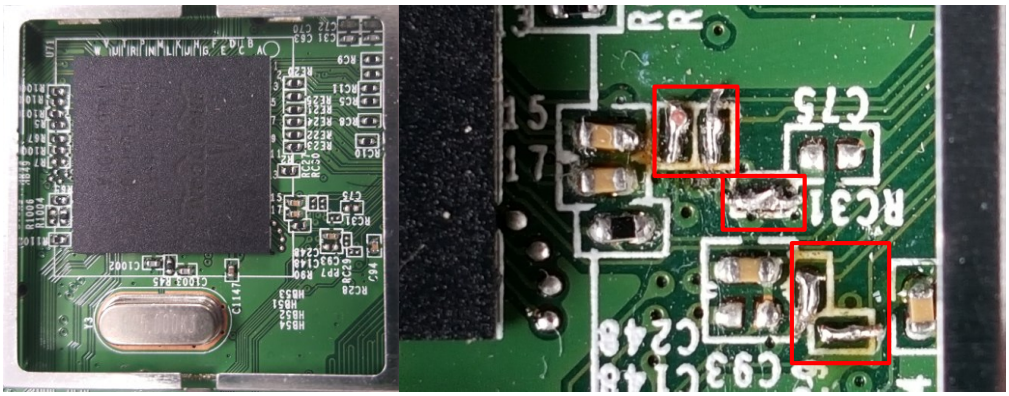


Figure 3: Bridged JTAG Connector

4.1 Serial Console, Part I

As a first step, we decided to keep it simple and started with the already functional UART connection to see what kind of information we get when hooking up a serial cable (3.3V to USB, like the ones from FTDI[8]).

[7] *http://www.grandideastudio.com/portfolio/jtagulator/*
[8] *http://www.ftdichip.com/Products/Cables/USBTTLSerial.htm*

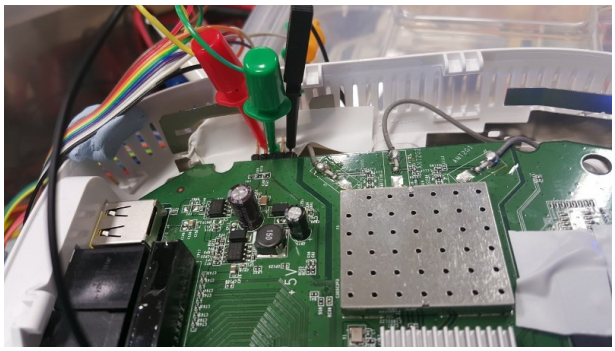


Figure 4: Connected Serial Cable

After doing so (as can be seen in the figure above) we've got a non-interactive shell where we could see the whole boot process of the access point.

4.1.1 Proof of Concept

The following section shows the output via UART while the AP was booting (for us interesting output is highlighted red):

```
System Bootstrap, Version CFE 5.10.144.16 Evora , RELEASE SOFTWARE
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 2010 by cisco Systems, Inc.
Build Date: Tue Dec 21 18:25:58 EST 2010 (root@localhost.localdomain)

Init Arena
Init Devs.
Boot partition size = 262144(0x40000)
et0: Broadcom BCM47XX 10/100/1000 Mbps Ethernet Controller 5.10.144.0
CPU type 0x19740: 480MHz
Tot mem: 65536 KBytes

CFE mem:    0x80700000 - 0x807A25F0 (665072)
Data:       0x80738810 - 0x8073B670 (11872)
BSS:        0x8073B670 - 0x8073C5F0 (3968)
Heap:       0x8073C5F0 - 0x807A05F0 (409600)
Stack:      0x807A05F0 - 0x807A25F0 (8192)
Text:       0x80700000 - 0x80738808 (231432)

Key = LOCKED !, Skipped memory test
[…}
Boot from golden image
Loader:raw Filesys:raw Dev:flash0.os File: Options:(null)
Loading: ... 1613824 bytes read
```

```
Entry at 0x80001000
[…]
Linux version 2.4.20 (aut@wnbu-bld-lnx-05) (gcc version 3.2.3 with Broadcom
modifications) #1 Sun Feb 27 23:11:50 PST 2011
[…]
Kernel command line: root=/dev/mtdblock2 noinitrd console=ttyS0,115200
CPU: BCM4716 rev 1 at 480 MHz
[…]
Flash device: 0x1000000 at 0x1c000000
Physically mapped flash: squash filesystem with lzma found at block 931
Creating 11 MTD partitions on "Physically mapped flash":
0x00000000-0x00040000 : "boot"
0x00040000-0x00440000 : "linux"
0x000e8cb8-0x00440000 : "rootfs"
0x00440000-0x00840000 : "app1"
0x00840000-0x00c40000 : "app2"
0x00c40000-0x00f80000 : "log"
0x00f80000-0x00fa0000 : "capwap"
0x00fa0000-0x00fc0000 : "df"
0x00fc0000-0x00fe0000 : "key"
0x00fe0000-0x01000000 : "nvram"
0x00040000-0x00440000 : "golden"
Found a 0MB  serial flash
[…}

Current Key state: 0x6c6f636b (LOCK)

Key state : LOCK

mv: unable to rename `/tmp/certs/self_signed_cert.cert': No such file or directory
.........
Certificates in /tmp.
WARNING: console log level set to 1
vlan1: Operation not supported
wlconf: unit1, country_code CN, ccode CN, regrev 5

[…]

CAPWAP Start, waiting for WAN IP address (0)
```

4.2 Serial Console, Part II

Gaining access to a non-interactive shell provided us with some information about the internals of the access point. As discussed in the kick-off workshop with the customer the assessment's objectives included eventually getting root access on the access point. Not knowing much about the device –

especially nothing about "CFE" the boot loader mentioned in the boot process above – we tried to interrupt the boot process by hitting simple key combinations like CTRL+C or ESC, which did not yield success.

4.3 NAND Glitching

As known from other embedded devices with flash memory, there are ways to gain access to a boot loader (e.g. "Das U-Boot[9]") by interrupting the correct startup of the embedded OS via grounding a data line of the flash at a specific point of time. This is called "NAND glitching". The reason for this approach is because by blocking access to the flash memory, the program logic boots into the boot loader when there is no OS image it could boot from, like a failsafe mechanism.

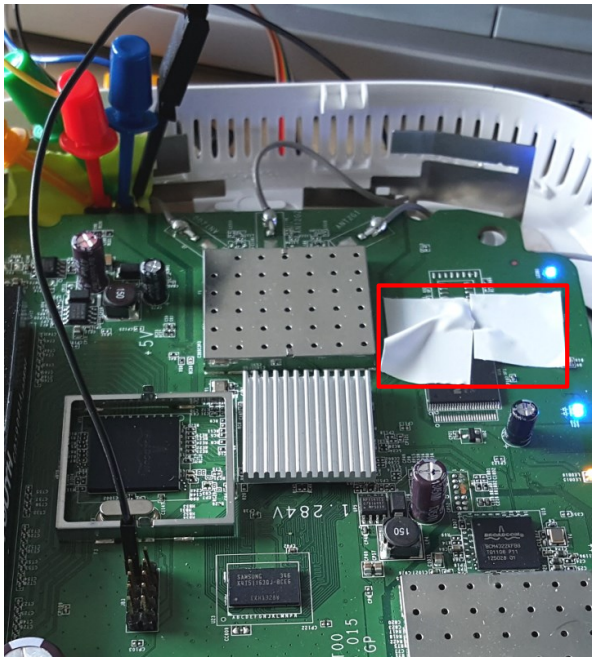
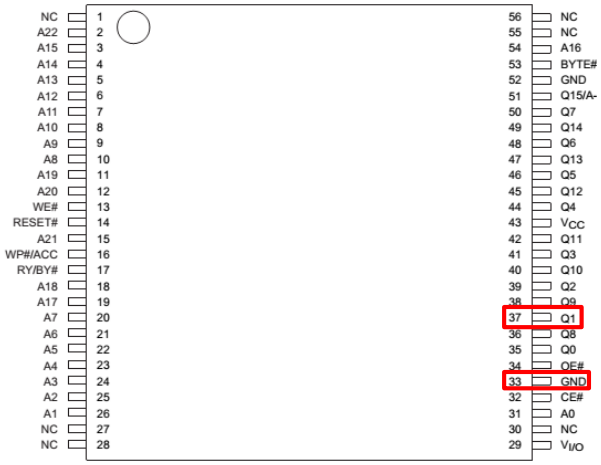


Figure 5: Grounding Wire and Isolated Pin at Flash Memory

To find out which pins are used for data and GND, the datasheet of the NAND flash [10] was looked at, in order to gather the knowledge of where to connect the wires.

[9] *http://www.denx.de/wiki/publish/U-Bootdoc/U-Bootdoc.pdf*
[10] *http://media.digikey.com/pdf/Data%20Sheets/Macronix/MX29GL128E.pdf*



PIN DESCRIPTION

SYMBOL	PIN NAME
A0~A22	Address Input
Q0~Q14	Data Inputs/Outputs
Q15/A-1	Q15(Word Mode)/LSB addr(Byte Mode)
CE#	Chip Enable Input
WE#	Write Enable Input
OE#	Output Enable Input
RESET#	Hardware Reset Pin, Active Low
WP#/ACC*	Hardware Write Protect/Programming Acceleration input
RY/BY#	Ready/Busy Output
BYTE#	Selects 8 bits or 16 bits mode
VCC	+3.0V single power supply
GND	Device Ground
NC	Pin Not Connected Internally
VI/O	Power Supply for Input/Output

Figure 6: NAND Flash Pinout

It took us a couple of freezes and reboots of the access point to figure out the correct time when to interrupt the connection, but finally we were able to do so! After a little bit of training we got it working on the first try in consecutive events. As can be seen from the output below, the interrupt had to be triggered while `Loading: ...` appeared on the screen.

```
Boot from golden image
Loader:raw Filesys:raw Dev:flash0.os File: Options:(null)
Loading: ... 1613824 bytes read
```

Otherwise there would be an error like `Golden image validation fail !`.

This attack could also be automated by using something like an Arduino[11] and a few lines of code.

4.3.1 Proof of Concept

Triggering the interrupt at the correct time resulted in the error message below and access to the interactive boot loader.

```
Boot from golden image
Loader:raw Filesys:raw Dev:flash0.os File: Options:(null)
Loading: .. 835584 bytes read
Failed.
Could not load flash0.os:: Error
CFE>
```

[11] *https://www.arduino.cc/*

4.4 CFE (Common Firmware Environment)

After gaining access to the shell, the first step was to identify potential commands the shell might accept.

The output below shows all accepted commands:

```
CFE> help
Available commands:

et                  Broadcom Ethernet utility.
nvram               NVRAM utility.
reboot              Reboot.
flash               Update a flash memory device
batch               Load a batch file into memory and execute it
go                  Verify and boot OS image.
boot                Load an executable file into memory and execute it
load                Load an executable file into memory without executing it
save                Save a region of memory to a remote file via TFTP
printdf             Print all parameters of DF Partition
mtest               memory test
add_nvram           add garbage nvram parameters
ctrlc               ctrlc test
getrevnum           get PCA Revision Number
setrevnum           set PCA Revision Number
getassemnum         get PCA Assembly Number
setassemnum         set PCA Assembly Number
getr1macblk         get Radio1 mac address block size
setr1macblk         set Radio1 mac address block size
getr1mac            get radio 1 mac address
setr1mac            set radio 1 mac address
getr0macblk         get Radio0 mac address block size
setr0macblk         set Radio0 mac address block size
getr0mac            get radio 0 mac address
setr0mac            set radio 0 mac address
getmacblk           get ethernet mac address block size
setmacblk           set ethernet mac address block size
getmac              get ethernet mac address
setmac              set ethernet mac address
getdevnum           get Deviation Number
setdevnum           set Deviation Number
getfabpartnum       get PCB Fab Part Number
setfabpartnum       set PCB Fab Part Number
getsn               get PCB serial number
setsn               set PCB serial number
getboardrev         get Board Revisoin
setboardrev         set board revision
getpartnum          get Part Number
setpartnum          set Part Number
ledoff              set gpio test
```

```
ledon               set gpio test
reset_det           detect reset button
getkey              read key value
setkey              Set key value to 0x4D464731
ping                Ping a remote IP host.
arp                 Display or modify the ARP Table
ifconfig            Configure the Ethernet interface
show clocks         Show current values of the clocks.
show devices        Display information about the installed devices.
unsetenv            Delete an environment variable.
printenv            Display the environment variables
setenv              Set an environment variable.
help                Obtain help for CFE commands

For more information about a command, enter 'help command-name'
*** command status = 0
```

By going through the list of commands, we identified potential interesting ones, namely setting environmental variables and setting some key values. It should be noted that during the boot process there was a line which mentions the following:

```
Current Key state: 0x6c6f636b (LOCK)
Key state : LOCK
```

We verified this by issuing the following command:

```
CFE> getkey
0x6C6F636B
```

Potentially something can be unlocked here, as `0x6c6f636b` is the hexadecimal representation of `"LOCK"` mentioned in the brackets right next to the key value. Additionally, the key value `0x4D464731` mentioned in the help context reveals `MFG1`. Since we knew nothing about possible key states and their effects, we took the risk and just entered `setkey` without any value or option.

```
CFE> setkey
Key write to INIT done!!
*** command status = 0
CFE> getkey
0x454E4731
*** command status = 0
```

This revealed a new key! `0x454E4731` which is `ENG1` in ASCII, which sounds like "`Engineering`" – and potentially revealing an engineering menu/mode. In order to take effect of the new key state we rebooted the access point. Following the boot process, we could see that now our newly set key is used:

```
Current Key state: 0x454e4731 (INIT)
Key state : INIT
```

Now, after the AP finished booting we simply hit Return and the device comes up with shell access to a `BusyBox` Linux. The changes are permanent, so access is available all the time. The detailed output of various commands can be found in the Appendix.

It's also possible to interrupt the boot of the OS to enter `CFE` by hitting `CTRL+C`, after the key value is set like here.

4.5 BusyBox

Among other things, it was possible to read and alter the actual configuration of the AP stored in the simulated `nvram`. Also stored in the `nvram` were different usernames and passwords in plaintext. As of the engagement's objectives one main question to answer was how the access point stored the Cisco Manufacturer Installed Certificate (MIC). As we now have full access, we crawled through the directories to see whether we can find something relevant.

4.6 Extracting the MIC Certificate from the Access Point

During the crawling, we could identify two potentially interesting directories on the access point, namely `/tmp/certs/` and `usr/sbin/certs/` as can be seen on the output below. The ones in `/tmp/certs` are looking promising as there is the root certificate of the Cisco Manufacturing CA as well as "some signed certificate" with the (presumably) corresponding private key.

```
# ls -la tmp/certs/
-rw-r--r--    1 0        0            1740 Jan  1 00:00 cisco-mfg-root-cert.cert
-rw-r--r--    1 0        0            1602 Jan  1 00:00 cisco_signed_cert.cert
-rw-r--r--    1 0        0            1679 Jan  1 00:00 priv_key
# ls -la usr/sbin/certs/
-rw-r--r--    1 0        0            1192 Feb 27  2011 cisco-root-cert.cert
-rw-r--r--    1 0        0            1456 Feb 27  2011 airespace-old-root-cert.cert
-rw-r--r--    1 0        0            1570 Feb 27  2011 airespace-new-root-cert.cert
-rw-r--r--    1 0        0            1619 Feb 27  2011 airespace-device-root-cert.cert
```

We extracted both certificates and used `openssl` to validate the content of the certificate `cisco_signed_cert.cert` as can be seen in the output below:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7a:61:08:f8:00:00:00:02:c4:ed
    Signature Algorithm: sha1WithRSAEncryption
```

```
        Issuer: O=Cisco Systems, CN=Cisco Manufacturing CA
        Validity
            Not Before: Feb  5 07:57:13 2014 GMT
            Not After : Feb  5 08:07:13 2024 GMT
        Subject: C=US, ST=California, L=San Jose, O=Cisco Systems, CN=OEAP602-
003A9AC17E00/emailAddress=support@cisco.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ae:c1:bc:81:0d:4e:c4:39:11:69:cc:e5:cb:c1:
                    74:f2:1a:2f:42:cd:ea:39:7c:99:16:eb:ab:27:4a:
                    12:df:1a:7a:46:04:c0:28:42:53:35:df:33:ea:4a:
                    d6:31:a9:70:f2:cf:bd:e9:a1:86:03:20:a4:ea:5a:
                    bf:30:a9:e5:e1:ca:64:d4:45:b9:06:40:49:2b:a1:
                    08:5e:e6:f4:a6:a9:2f:89:b5:80:04:b4:a2:d7:85:
                    b4:0c:bf:69:bb:c8:2f:9e:20:71:af:13:54:ee:5e:
                    c5:10:d6:5e:3a:8b:7b:3a:b5:53:fe:00:66:55:41:
                    cf:7b:c8:ce:95:81:55:50:a2:a0:ed:58:1e:f2:59:
                    98:19:f3:a3:81:3f:a8:8c:34:a9:13:5a:27:af:f7:
                    2a:e8:a8:f9:70:7e:85:84:25:1a:db:4a:f2:4d:07:
                    2b:a5:61:a9:32:02:30:b3:fa:7e:8d:a1:68:46:bf:
                    c7:d8:f9:42:b2:1d:00:f6:95:dd:0d:a2:66:13:86:
                    f4:52:22:3c:2f:a5:cd:f2:5f:59:af:b5:96:65:eb:
                    39:1d:7c:62:88:49:3c:42:dd:c4:9a:af:d4:9a:fe:
                    8a:7b:ba:ab:64:02:a0:de:c0:25:64:9d:22:9a:23:
                    29:5b:35:39:e3:0b:0f:ba:b7:a5:ab:c3:13:2d:44:
                    cf:5f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Subject Key Identifier:
                51:79:61:F6:7D:15:98:7A:9A:E8:06:CE:CE:06:50:AA:A8:78:1E:8D
            X509v3 Authority Key Identifier:
                keyid:D0:C5:22:26:AB:4F:46:60:EC:AE:05:91:C7:DC:5A:D1:B0:47:F7:6C

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://www.cisco.com/security/pki/crl/cmca.crl

            Authority Information Access:
                CA Issuers - URI:http://www.cisco.com/security/pki/certs/cmca.cer

            1.3.6.1.4.1.311.20.2:
                .0.I.P.S.E.C.I.n.t.e.r.m.e.d.i.a.t.e.O.f.f.l.i.n.e
    Signature Algorithm: sha1WithRSAEncryption
```

```
0c:64:a9:ee:42:bb:5b:e6:62:16:16:88:c0:8f:08:b0:44:03:
ac:fe:ca:ae:8e:b3:2b:58:91:5d:54:c4:45:5f:dc:fa:42:39:
01:76:83:04:28:31:9b:67:63:1f:65:0f:f5:78:f1:d2:c2:f7:
f9:1c:e5:70:47:d8:d4:fb:1a:c8:a2:51:7a:29:99:4a:7c:88:
20:be:70:e4:d1:b8:f5:4e:1f:48:94:57:e6:e9:3d:e1:cb:c5:
c3:21:cc:6f:d9:eb:eb:3f:cc:00:4e:77:e7:76:bc:8a:99:47:
ec:0e:64:56:ab:10:e7:85:95:8a:25:96:3a:81:16:fd:ba:e9:
26:2e:7f:a0:6f:49:44:89:a6:cd:a0:ab:1d:be:37:cd:69:7b:
54:28:51:fd:6b:8e:7c:ba:9a:33:e7:d8:9d:c5:ae:3d:a3:07:
79:50:1b:25:80:e1:c9:a2:f7:e8:91:a3:21:d5:99:27:2d:0c:
bf:8b:d4:d3:8d:8b:4e:c0:f2:ef:67:0b:9e:1c:d9:c4:cb:fd:
c3:24:86:64:d6:fd:81:28:16:95:7c:c1:47:33:1c:59:d5:21:
90:06:01:9e:bc:c1:67:93:ea:93:30:6c:9e:d7:8b:e4:f9:bb:
04:ec:a4:9e:d3:3b:8b:cb:e8:38:2d:d2:34:c9:85:c6:8c:27:
37:36:17:87
```

This is the certificate of the access point signed from the Cisco Manufacturing CA. The next step was comparing the certificate and the private key. By comparing the modules of the certificate and the private key, it was possible to match it with the `cisco_signed_cert.cert`. We now have verified that it is in fact the private key corresponding to the certificate, and we were able to easily extract both from the filesystem.

```
$ openssl x509 -noout -modulus -in cisco_signed_cert.cert | openssl md5
(stdin)= b46b8d168991c72fa3de555eaa6c6420
$ openssl rsa -noout -modulus -in priv_key | openssl md5
(stdin)= b46b8d168991c72fa3de555eaa6c6420
```

4.7 JTAG

As it was not possible to set the provided device into a working state anymore after the command `setkey` was used, we had to look for another way getting it working again. `Setkey` seems to be a "one way thing" which doesn't accept any parameters.

As mentioned earlier, we already made JTAG useable by connecting the open pads.

By using a JTAG debugger, the firmware recovery script of the software OpenOCD[12] and some custom board file with the correct memory addresses (which we captured via UART), it was possible to dump, alter and flash the memory of the access point.

Partition layout / memory addresses:

```
0x00000000-0x00040000 : "boot"
```

[12] *http://openocd.org*

```
0x00040000-0x00440000 : "linux"
0x000e8cb8-0x00440000 : "rootfs"
0x00440000-0x00840000 : "app1"
0x00840000-0x00c40000 : "app2"
0x00c40000-0x00f80000 : "log"
0x00f80000-0x00fa0000 : "capwap"
0x00fa0000-0x00fc0000 : "df"
0x00fc0000-0x00fe0000 : "key"
0x00fe0000-0x01000000 : "nvram"
0x00040000-0x00440000 : "golden"
```

Following a part of the "`key`"-partition in the "init" or "engineering" state:

```
$ xxd key.bin
00000000: 3147 4e45 ffff ffff ffff ffff ffff ffff  1GNE............
00000010: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000040: ffff ffff ffff ffff ffff ffff ffff ffff  ................
```

By simply changing "`1ENG`" back to "`kcol`" (the original one) the device will start working again:

```
$ xxd key.bin
00000000: 6b63 6f6c ffff ffff ffff ffff ffff ffff  kcol............
00000010: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000040: ffff ffff ffff ffff ffff ffff ffff ffff  ................
```

Furthermore, it was possible to exchange the certificates and keys stored on the device.

5 Conclusion & Outlook

We have demonstrated some quick options to access the memory of the assessed device. There is no high complexity behind these attacks, but the security impact can be very high. Operating system files including certificates and encryption material have been dumped and extracted impacting the integrity of the access point and potentially the confidentiality of the transmitted information.

Still, in this case the product is End of Sale and in End of Life but as the assessment started as a project initiated from a customer, this shows that the device is currently used in corporate environments.

Yet again, this shows that there is a need to improve hardware design regarding to its security and perform security assessments to reveal its vulnerabilities.

6 Appendix

6.1 Disclosure Timeline

02.05.2016	Mailed Cisco PSIRT
02.05.2016	First reaction from Cisco
07.06.2016	Finally sent the report to PSIRT (lot of trouble sending the document)
29.06.2016	Info that they'll block UART on boot, but can't disable JTAG w/o "physical changes", also no crypto possible; Announcement of bootloader update
22.09.2016	Case made public by Cisco; CVSSv2 rating too low for Security Advisory because product is EoS and EoL (as it seems)

6.2 Device Information

```
BusyBox v0.60.0 (2011.02.28-07:10+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.
# pwd
/
# busybox
BusyBox v0.60.0 (2011.02.28-07:10+0000) multi-call binary

Usage: busybox [function] [arguments]...
   or: [function] [arguments]...

        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use, and BusyBox
        will act like whatever it was invoked as.

Currently defined functions:
        busybox, cat, chmod, cp, date, dd, echo, free, grep, gunzip, gzip,
        ifconfig, insmod, kill, killall, klogd, ln, ls, lsmod, mkdir,
        mknod, more, mount, msh, mv, ping, ps, pwd, reboot, rm, rmdir,
        rmmod, route, sh, sleep, syslogd, tar, tftp, top, umount, wget,
        zcat

# ls -la /us# ls -la /usr/bin# ls -la /usr/bin/
lrwxrwxrwx    1 0        0               17 Feb 27  2011 wget -> ../../bin/busybox
lrwxrwxrwx    1 0        0               17 Feb 27  2011 top -> ../../bin/busybox
lrwxrwxrwx    1 0        0               17 Feb 27  2011 tftp -> ../../bin/busybox
lrwxrwxrwx    1 0        0               17 Feb 27  2011 killall -> ../../bin/busybox
lrwxrwxrwx    1 0        0               17 Feb 27  2011 free -> ../../bin/busybox
drwxr-xr-x    5 0        0               45 Feb 27  2011 ..
drwxr-xr-x    2 0        0               59 Feb 27  2011 .
# # ls -la /usr/bin/        bin# ls -la /bin/
lrwxrwxrwx    1 0        0                7 Feb 27  2011 zcat -> busybox
-rwxr-xr-x    1 0        0           244288 Feb 27  2011 wps_monitor
lrwxrwxrwx    1 0        0                7 Feb 27  2011 umount -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 tar -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 sleep -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 sh -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 rmdir -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 rm -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 pwd -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 ps -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 ping -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 mv -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 msh -> busybox
lrwxrwxrwx    1 0        0                7 Feb 27  2011 mount -> busybox
```

```
lrwxrwxrwx    1 0        0               7 Feb 27  2011 more -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 mknod -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 mkdir -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 ls -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 ln -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 kill -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 gzip -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 gunzip -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 grep -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 echo -> busybox
-rwxr-xr-x    1 0        0           61856 Feb 27  2011 eapd
lrwxrwxrwx    1 0        0               7 Feb 27  2011 dd -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 date -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 cp -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 chmod -> busybox
lrwxrwxrwx    1 0        0               7 Feb 27  2011 cat -> busybox
-rwxr-xr-x    1 0        0          350852 Feb 27  2011 busybox
drwxr-xr-x   13 0        0             120 Feb 27  2011 ..
drwxr-xr-x    2 0        0             291 Feb 27  2011 .

#ls -la /sbin/
lrwxrwxrwx    1 0        0               2 Feb 27  2011 write -> rc
lrwxrwxrwx    1 0        0              14 Feb 27  2011 syslogd -> ../bin/busybox
lrwxrwxrwx    1 0        0               2 Feb 27  2011 stats -> rc
lrwxrwxrwx    1 0        0               2 Feb 27  2011 showeventlog -> rc
lrwxrwxrwx    1 0        0              14 Feb 27  2011 route -> ../bin/busybox
lrwxrwxrwx    1 0        0              14 Feb 27  2011 rmmod -> ../bin/busybox
lrwxrwxrwx    1 0        0              14 Feb 27  2011 reboot -> ../bin/busybox
-rwxr-xr-x    1 0        0          204892 Feb 27  2011 rc
lrwxrwxrwx    1 0        0               2 Feb 27  2011 preinit -> rc
lrwxrwxrwx    1 0        0              14 Feb 27  2011 lsmod -> ../bin/busybox
lrwxrwxrwx    1 0        0              14 Feb 27  2011 klogd -> ../bin/busybox
lrwxrwxrwx    1 0        0              14 Feb 27  2011 insmod -> ../bin/busybox
lrwxrwxrwx    1 0        0               2 Feb 27  2011 init -> rc
lrwxrwxrwx    1 0        0              14 Feb 27  2011 ifconfig -> ../bin/busybox
lrwxrwxrwx    1 0        0               2 Feb 27  2011 hotplug -> rc
lrwxrwxrwx    1 0        0               2 Feb 27  2011 erase -> rc
lrwxrwxrwx    1 0        0               2 Feb 27  2011 df -> rc
lrwxrwxrwx    1 0        0               2 Feb 27  2011 cert -> rc
lrwxrwxrwx    1 0        0               2 Feb 27  2011 bootupgrade -> rc
drwxr-xr-x   13 0        0             120 Feb 27  2011 ..
drwxr-xr-x    2 0        0             218 Feb 27  2011 .
# ls -la
drwxr-xr-x    2 0        0             287 Feb 27  2011 www
lrwxrwxrwx    1 0        0               7 Feb 27  2011 var -> tmp/var
drwxr-xr-x    5 0        0              45 Feb 27  2011 usr
drwxr-xr-x    1 0        0               0 Dec 31  1999 tmp
drwxr-xr-x    2 0        0               3 Feb 27  2011 sys
```

```
drwxr-xr-x    2 0        0             218 Feb 27  2011 sbin
dr-xr-xr-x   35 0        0               0 Dec 31  1999 proc
drwxr-xr-x    2 0        0               3 Feb 27  2011 mnt
lrwxrwxrwx    1 0        0               9 Feb 27  2011 media -> tmp/media
drwxr-xr-x    3 0        0             124 Feb 27  2011 lib
drwxr-xr-x    2 0        0              43 Feb 27  2011 etc
drwxr-xr-x    1 0        0               0 Dec 31 16:00 dev
drwxr-xr-x    2 0        0             291 Feb 27  2011 bin
drwxr-xr-x   13 0        0             120 Feb 27  2011 ..
drwxr-xr-x   13 0        0             120 Feb 27  2011 .

#mount
rootfs on / type rootfs (rw)
/dev/root on / type squashfs (ro)
none on /dev type devfs (rw)
proc on /proc type proc (rw)
ramfs on /tmp type ramfs (rw)

# df show
Part Number                          :     74-8383-01
Board Revision                       :     1216
PCB Serial Number                    :     FOC18025T9N
PCB Fab Part Number                  :     308.00855.015
Deviation Number                     :     0
Base MAC Address                     :     00:3A:9A:C1:7E:00
MAC Address Block Size               :     8
Radio 0 MAC Address                  :     00:3a:9b:02:17:c0
Radio 0 MAC Address Block Size       :     16
Radio 1 MAC Address                  :     00:3a:9b:02:17:d0
Radio 1 MAC Address Block Size       :     16
PCA Assembly Number                  :     74-8383-01
PCA Revision Number                  :     1216
Product/Model Number                 :     AIR-OEAP602I-E-K9
Top Assembly Part Number             :     800-36713-01
Top Revision Number                  :     A0
Top Assembly Serial Number           :     FCZ1808Y01J
RMA Test History                     :
RMA History                          :
RMA Number                           :
Device Type                          :
Max Association Allowed              :
Radio(2.4G) Carrier Set              :     0001
Radio(2.4G) Max Transmit Power Level :     0
Radio(2.4G) Antenna Diversity Support :
Radio(2.4G) Encryption Ability       :
Radio(5G) Carrier Set                :     000C
Radio(5G) Max Transmit Power Level   :     0
Radio(5G) Antenna Diversity Support  :
```

```
Radio(5G) Encryption Ability          :
Radio(802.11g) Radio Mode             :
PEP Product Identifier (PID)          :     AIR-OEAP602I-E-K9
PEP Version Identifier (VID)          :     V01
#
ifconfig
br0       Link encap:Ethernet  HWaddr 00:3A:9A:C1:7E:00
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:65 errors:0 dropped:0 overruns:0 frame:0
          TX packets:63 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5248 (5.1 kb)  TX bytes:28273 (27.6 kb)

br1       Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:10.0.2.1  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

eth0      Link encap:Ethernet  HWaddr 00:3A:9A:C1:7E:00
          UP BROADCAST RUNNING ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:67 errors:0 dropped:0 overruns:0 frame:0
          TX packets:333 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:7366 (7.1 kb)  TX bytes:84697 (82.7 kb)
          Interrupt:4 Base address:0x2000

eth1      Link encap:Ethernet  HWaddr 00:3A:9B:02:17:C7
          UP BROADCAST RUNNING ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:46876
          TX packets:0 errors:245 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:3 Base address:0x1000

eth2      Link encap:Ethernet  HWaddr 00:3A:9B:02:17:C8
          UP BROADCAST RUNNING ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:25279
          TX packets:0 errors:249 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:6 Base address:0x8000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

```
          UP LOOPBACK RUNNING MULTICAST  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

vlan1     Link encap:Ethernet  HWaddr 00:3A:9A:C1:7E:00
          UP BROADCAST RUNNING ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:67 errors:0 dropped:0 overruns:0 frame:0
          TX packets:264 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5900 (5.7 kb)  TX bytes:41389 (40.4 kb)
```

6.3 Disclaimer