Eight ou two eleven

Dynamic inspection of Broadcom Wi-Fi cards on mobile devices

Matias Eissler – HITBAMS2015
Agenda

• Overview of Broadcom Wi-Fi NiC mobile devices
  – Architecture
  – Attack surface & possibilities

• Tool:
  – Dynamic inspection.
  – Why not just make a debugger?
  – Our objective
  – Explore findings along the way.

• Usage of the tool to inspect firmware
Previous works

• Much has been done on network card firmware. See Triulzi[1], Delugré[2], others [3]
• Mobile devices
  – Firmware modified for monitor mode and raw injection on iOS & Android by two different teams (Andres Blanco, bcmmon team)
  – Vulnerabilities discovered: CVE-2012-2619
  – Not much (public) research after that.
Broadcom huge WI-FI player
What do the cards look like?
What’s inside? CPU, memory and cores

HOST (Mobile) --> SDIO --> Cortex M3/R4 --> SRAM --> d11 --> PHY --> Mac

Air

Chip Common
Attack surface & possibilities

• 802.11 implementation bug -> RCE Firmware
  – Pivot Firmware -> Driver
  – Man-in-the-middle to inject browser/app exploits
  – At least pivot to a target LAN:
Even more surface

• Firmware supports wide range of features:
  – TCP
  – ICMP & ARP offloading
  – Firewall implementation
  – Mobile hotspot, Wi-Fi Direct, AirDrop
  – Proprietary 802.11 extensions (Broadcom/Cisco)

• We need to play more with these firmwares!
Mobile products timeline

- **i900 Omnia** released July, 2008
- **M7600 Beat DJ** i8910 Omnia HD released May, 2009
- **i9000 Galaxy S** released June, 2010
- **Vibrant T959** released July, 2010
- **i8510 INNOV8** released Sept, 2008
- **Jet S8000** released June, 2009
- **i5700 Galaxy Spica** released Nov, 2009
- **Sunburst A697** released March, 2010
- **Mesmerize i500** released Oct, 2010
- **Galaxy S 4G T959V** Gem I100 released Feb, 2011
- **Galaxy S II T989** released Oct, 2011

- **iPhone 3G** released July 11, 2008
- **iPhone 3GS** released June 19, 2009
- **iPhone 4** released June 24, 2010
- **iPhone 4S** released Oct 14, 2011
Very soon you end up buried in a sea of devices
Objectives

- Dynamically inspect firmware
- Be as OS/Device independent as possible
Why dynamic?

• Static inspection only gets you that far.
• Once you have all memory dumped, understanding everything from a static perspective is limited. E.g. indirect calls.
• If you manage to get a crash it is hard to understand what happened.
Firmware is Separated in two regions

Region 1

Region 2

Loaded from filesystem:
Only protected by CRC
/etc/wifi/firmware
/usr/share/firmware/wifi

Burned into ROM:
Not initially accessible to us
Communication

Device CPU → Kernel

Card CPU → Firmware

IOCTL

SDIO
Communication

Device CPU

Kernel

Firmware

Card CPU

Users pace

IOCTL message over SDIO

Apple: IOC_CARD_SPECIFIC

Android: SIOCDEVPRIVATE
Proposed solution

- If we modify the firmware to support new IOCTL msgs: Read & Write.
- Send a user -> kernel IOCTL, that encapsulates a Kernel -> firmware IOCTL
- If we can do this, then we can even write python code, from userspace, that will read and write memory from the firmware!
Identifying IOCTL Handler

- Search for switch with lots of cases.
- Or search for WLC_MAGIC IOCTL=0x14e46c77
- Sometimes the handler is on Region 2... BUT if we have an earlier or different firmware we can find the caller.
- If all else fails, follow interrupt handler path
Typical hooking

String

Original Code  Point of interest  Our code
```
B1 F5 7A 4F
05 D0
B1 F5 7B 4F
06 D0
07 46
0E 46
70 47

; ---------

read

10 46
11 68
52 68
03 E0

; ---------

write

10 68
12 F1 08 01
52 68

done

02 4B
98 47
00 20
BD E8 FC 81

; ---------

CMP.W R1, #0xFA00
BEQ read
CMP.W R1, #0xFB00
BEQ write
MOV R7, R0
MOV R6, R1
BX LR

MOV R0, R2
LDR R1, [R2]
LDR R2, [R2,#4]
B done

LDR R0, [R2]
ADDS.W R1, R2, #8
LDR R2, [R2,#4]

LDR R3, =memcpy+1
BLX R3 ; memcpy
MOVS R0, #0
POP.W {R2-R8,PC}
```
R&W Little Demo
R&W Little Demo
Read & Write. Now what?

• Dump Region 2.
• At this point we can read & write to memory mapped registers
• All sort of counters, stats, even packets.
• Most importantly we can modify the code.
  – And we can do that without having to create new firmwares each time!
def createHook(self, pointCode):
    code = ('
    "00BF"  # NOP ; placeholder to place the instructions smashed by the jmp
    "00BF"  # NOP ; that the tracer injected.
    "07B4"  # PUSH {R0-R2}
    "00BF"  # NOP ; placeholder to place a mov instruction with the desired register.
    "0449"  # LDR R1, =sub_22CA0
    "0A68"  # LDR R2, [R1]
    "102A"  # CMP R2, #0x10
    "02D0"  # BEQ done
    "0432"  # ADDS R2, #4
    "0A60"  # STR R2, [R1]
    "8850"  # STR R0, [R1,R2]
    # done
    "07BC"  # POP {R0-R2}
    "7047"  # BX LR
    "0000"  # align
    # "A02C0200"
    ).decode('hex')
    code += struct.pack("<L", self.DataAddr)

    code = code.replace("\x00\xbf\x00\xbf", pointCode)
    code = code.replace("\x00\xbf", self.assembleMov())
    return code
First Tracer

• Given an address and a register:
  – Create hook & hook handler code.
  – Clear a storage area
  – The read from storage
  – Usage as simple as:

```python
t = Tracer(0x026CB4, 'R3')
t.hook()
try:
    while True:
        print t.traces()
        time.sleep(1)
except:
    t.unhook()
```
What about region 2?
What about region 2?

- Enter flash patch
  - Set up a remap table
  - Comparators
  - Enable FPB through a control register.

- Basically, it is like we are setting up the MMU to modify instructions on fetch.
Flash patch operation diagram
Tracer again

• Setup Hook handler as before and then:
  – Write remap table in memory
  – Setup comparators
  – Enable FPB
  – Houston: we have tracepoints (kindda).
Wait a minute!

- Basically, it is like we are setting up the MMU to modify instructions on fetch.
How does it work?
Non-persistent rootkit?

• Scenario:
  – Compromised device.
  – Modifies Region 1 file on disk.
  – Loads into the card.
  – Restores Region 1 file.
  – Exfiltrate traffic or pivot through another network, side-channel, etc.
Want even more stealth?

• Make it so that even if someone can read the firmware live from the card memory. It can't!
• Setup remap table so that malicious code is hidden.
• What about the remap table? No problem! Remapping the remap table works!
100% Stealth?

• Answer is no:
  – Can’t remap control or comparator registers.
  – Have a limited number of comparator and remap entries.
  – If remap control register is disabled the whole deception falls.

• Still more work to discover hidden code.
Back to our tool

• Brief 802.11 review:
  – 3 Types of frames:
    • Data
    • Management
    • Control
  – Mgmt frames contain Information Elements

![Element format diagram](figure7-37)
Usual association process (management frames)

Installing the PTK, and where applicable the GTK keys, causes the MAC to encrypt and decrypt all subsequent MSDUs irrespective of their path through the controlled or uncontrolled ports.

Upon successful completion of the 4-Way Handshake, the Authenticator and Supplicant have authenticated each other; and the IEEE 802.1X Controlled Ports are unblocked to permit general data traffic. See Figure 5-13.
Association response

IEEE 802.11 Association Response, Flags: ........C

IEEE 802.11 wireless LAN management frame

- Fixed parameters (6 bytes)
- Tagged parameters (151 bytes)
  - Tag: Supported Rates 1(B), 2(B), 5.5(B), 11(B), 9, 18, 36, 54, [Mbit/sec]
  - Tag: Vendor Specific: Microsoft: WMM/WME: Parameter Element
    - Tag Number: Vendor Specific (221)
    - Tag length: 24
    - OUI: 00-50-f2 (Microsoft)
    - Vendor Specific OUI Type: 2
    - Type: WMM/WME (0x02)
    - WME Subtype: Parameter Element (1)
    - WME Version: 1
    - WME QoS Info: 0x00
      - Reserved: 00
    - Ac Parameters ACI 0 (Best Effort), ACM no, AIFSN 3, ECWmin 4, ECWmax 10, TXOP 0
    - Ac Parameters ACI 1 (Background), ACM no, AIFSN 7, ECWmin 4, ECWmax 10, TXOP 0
    - Ac Parameters ACI 2 (Video), ACM no, AIFSN 2, ECWmin 3, ECWmax 4, TXOP 94
    - Ac Parameters ACI 3 (Voice), ACM no, AIFSN 2, ECWmin 2, ECWmax 3, TXOP 47
Code processing association response

```
00026C9E  D5 F8 18 33  LDR.W  R3, [R5,#0x318]
00026CA2  72 68                                      LDR  R2, [R6,#4]
00026CA4  D5 F8 7C C5  LDR.W  R12, [R5,#0x57C]
00026CA8  06 93                                      STR  R3, [SP,#0x58+var_40]
00026CAA  42 F0 40 02  ORR.W  R2, R2, #0x40
00026CAE  0A 9B                                      LDR  R3, [SP,#0x58+var_30]
00026CB0  72 60                                      STR  R2, [R6,#4]
00026CB2  5A 78                                      LDRB  R2, [R3,#1]
00026CB4  0C F1 0E 00  ADD.W  R0, R12, #0xE
00026CB8  99 1C                                      ADDS  R1, R3, #2
00026CBA  CD F8 20 C0  STR.W  R12, [SP,#0x58+var_38]
00026CBE  E5 F3 AD F3  BL.W  memcpy
00026CC2  DD F8 20 C0  LDR.W  R12, [SP,#0x58+var_38]
00026CC6  9C F9 14 20  LDRSB.W  R2, [R12,#0x14]
00026CCA  00 2A                                      CMP  R2, #0
00026CCC  07 DA                                      BGE  loc_26CDE
```
Hook trace demo
THANKS!