Ghost in the Wireless, iwlwifi Edition

Gabriel Campana, Nicolas Iooss – SSTIC 2022
Context

- Up-to-date Ubuntu 18.04 LTS (Feb 2021)
- Android smartphone
- python -m http.server
# dmesg | tail
iwlwifi 0000:01:00.0: Start IWL Error Log Dump:
iwlwifi 0000:01:00.0: Status: 0x00000100, count: 6
iwlwifi 0000:01:00.0: Loaded firmware version: 34.0.1
iwlwifi 0000:01:00.0: 0x00000038 | BAD_COMMAND

...  
iwlwifi 0000:01:00.0: Start IWL Error Log Dump:
iwlwifi 0000:01:00.0: Status: 0x00000100, count: 7
iwlwifi 0000:01:00.0: 0x000000070 | ADVANCED_SYSASSERT

...  
iwlwifi 0000:01:00.0: 0x004F01A7 | last host cmd
ieee80211 phy0: Hardware restart was requested
Studied Wi-Fi chips

Intel Wireless-AC 8260

Intel Wireless-AC 9560
(Picture of a Companion RF Module)
Agenda

- The firmware & talking to the chip
- Vulnerability research
- Dynamic analysis experiments
- DMA through the paging memory
The Firmware
Intel WireLess (IWL) Wi-Fi on Linux
# Firmware file (for Intel Wireless for Linux)

```bash
# dmesg
iwlwifi 0000:00:14.3: loaded firmware version 46.6f9f215c.0
9000-pu-b0-jf-b0-46.ucode op_mode iwlmvm
```

```bash
# ls /lib/firmware/iwlwifi-9000-pu-b0-jf-b0-*
/lib/firmware/iwlwifi-9000-pu-b0-jf-b0-33.ucode
/lib/firmware/iwlwifi-9000-pu-b0-jf-b0-34.ucode
/lib/firmware/iwlwifi-9000-pu-b0-jf-b0-38.ucode
/lib/firmware/iwlwifi-9000-pu-b0-jf-b0-41.ucode
/lib/firmware/iwlwifi-9000-pu-b0-jf-b0-43.ucode
/lib/firmware/iwlwifi-9000-pu-b0-jf-b0-46.ucode
```

iwlwifi chooses a compatible firmware file using the API version

Firmware file format

Header:
- API version 0x2e = 46
- build number 6f9f215c

Entries:
- Type, Length, Value

No encryption
Firmware file format

drivers/net/wireless/intel/iwlwifi/fw/file.h

No encryption
$ parse_intel_wifi_fw.py iwlwifi-9000-pu-b0-jf-b0-46.ucode
- DEF_CALIB (12 bytes): ucode_type=REGULAR flow_trigger=0x0F0615DB event_trigger=0x0F02958B
- FW_VERSION (12 bytes): 46.6f9f215c.0
- LMAC_DEBUG_ADDRS (32 bytes):
  error_event_table_ptr = 0x00813C14
  log_event_table_ptr = 0x0046747C
...
- NUM_OF_CPU (4 bytes): 2
- SEC_RT (700 bytes): runtime microcode at 00404000..004042b8 (ACM Header)

No encryption
2 Processors?!?

- UMAC CPU (Upper Medium Access Controller)
- LMAC CPU (Lower Medium Access Controller)
- Physical Interface (Antennae)
- On-Chip Memory (SRAM, DCCM...)
- Wi-Fi chip

Main System

PCIe

Firmware loading

Unknown bus
Firmware memory layout

Firmware File
- DEF_CALIB
- FW_VERSION
- LMAC_DEBUG_ADDRS
  - NUM_OF_CPU 2
    - SEC_RT 00404000
    - SEC_RT 00800000
    - SEC_RT 00000000
    - SEC_RT 00456000
  ...
    - SEC_RT 00405000
    - SEC_RT c0080000
    - SEC_RT c0880000
    - SEC_RT 80448000
    ...

Wi-Fi chip
Memory
- 00000000..0037fff (229376 bytes)
- c008000..c008fff (65536 bytes)
- 00404000..004042b7 (696 bytes)
- 00405000..004052b7 (696 bytes)
- 80448000..80455ad3 (56020 bytes)
- 00456000..0048d873 (227444 bytes)
- 00800000..00817fff (98304 bytes)
- c0880000..c0887fff (32768 bytes)

Authenticated Module
Header:
- RSA-2048 public key
- Signature

cpu_rec: ARcompact

Linux: net/wireless/intel/iwlwifi/iwl-drv.c
#define FW_ADDR_CACHE_CONTROL 0xC0000000
Reverse all the things!

Tools: objdump, IDA Pro, Ghidra and custom Python scripts
Trying to modify the firmware (and the public key)

```
# dmesg
iwlwifi 0000:00:14.3: SecBoot CPU1 Status : 0x3030003, CPU2 Status: 0x0
```

FAIL

CHALLENGE ACCEPTED
Talking to the Chip
Linux Debug Filesystem

Memory read: almost anywhere :) (not 0048f000...0048ffff)

```bash
# DBGFS=/sys/kernel/debug/iwlwifi/0000:00:14.3
# cat $DBGFS/iwlvmfw/fw_ver
FW prefix: iwlwifi-9000-pu-b0-jf-b0-
FW: release/core43::6f9f215c
Device: Intel(R) Wireless-AC 9560 160MHz
Bus: pci

# dd if=$DBGFS/iwlvmfw/mem bs=1 count=128 | xxd
00000000: 2020 800f 0000 4000 2020 800f 0300 e474 ....@
00000010: 2020 800f 0300 3837 2020 800f 0000 c819 ....87
00000020: 6920 0000 6920 4000 6920 0000 6920 4000 i ..i @i ..i @.
00000030: 2020 800f 4700 14b6 6920 0000 6920 4000 ..G..i ..i @.
00000040: 6920 0000 4a20 0000 4a21 0000 4a22 0000 i ..J ..J!..J"
00000050: 4a23 0000 4a24 0000 4a25 0000 4a26 0000 J#..J$..J%..J&..
00000060: 4a27 0000 4a20 0010 4a21 0010 4a22 0010 J'..J ..J!..J"
00000070: 4a23 0010 4a24 0010 4a25 0010 4a26 0010 J#..J$..J%..J&..
```

Memory read: almost anywhere :) (not 0048f000...0048ffff)
Getting the PC (Program Counter)

```plaintext
// Linux: drivers/net/wireless/intel/iwlwifi/iwl-prph.h
#define UREG_UMAC_CURRENT_PC 0xa05c18
#define UREG_LMAC1_CURRENT_PC 0xa05c1c
#define UREG_LMAC2_CURRENT_PC 0xa05c20
```

```
# echo 0xa05c18 > $DBGFS/iwlmvm/prph_reg
# cat $DBGFS/iwlmvm/prph_reg
Reg 0xa05c18: (0xc0084f40)

# echo 0xa05c1c > $DBGFS/iwlmvm/prph_reg
# cat $DBGFS/iwlmvm/prph_reg
Reg 0xa05c1c: (0xb552)

# echo 0xa05c20 > $DBGFS/iwlmvm/prph_reg
# cat $DBGFS/iwlmvm/prph_reg
Reg 0xa05c20: (0x0)
```

UMAC pc

LMAC pc

No second LMAC

HOW‽
The perspective from iwlwifi (Linux)

More details in the article...
Arbitrary code execution
Host commands

- Communication with the chip through PCIe
- Commands processed by UMAC
- Undocumented commands
Vulnerability

```assembly
CODE:C0087A58 sub_C0087A58: # DATA XREF: data:LEGACY_GROUP4o
CODE:C0087A58
CODE:C0087A58
CODE:C0087A58
CODE:C0087A58
CODE:C0087A58
CODE:C0087A58
CODE:C0087A58
CODE:C0087A58

size       = -0x6C
flag       = -0x68
buffer     = -0x64
var_14     = -0x14
var_10     = -0x10

CODE:C0087A58 E1 C5
CODE:C0087A5A E1 C6
CODE:C0087A5C F1 C0
CODE:C0087A5E B8 C1
CODE:C0087A60 08 75
CODE:C0087A62 88 70
CODE:C0087A64 02 D9
CODE:C0087A66 00 DE
CODE:C0087A68 C9 72

push       r13
push       r14
push       blink
sub         sp, sp, 0x60 # ``
mov         r13, r0
mov r0, sp  # buffer
mov r1, 2   # count
mov r14, 0
mov r2, r14
jl          umac_fifo_read_bytes # read 2 dwords
ld          r1, [sp, 0x6C+size] # count
cmp r1, 0
beq  loc_C0087A84
add r0, sp, 0x6C+buffer # buffer
mov r2, r14
jl          umac_fifo_read_bytes # read `count` dwords
```
Exploitation

Attention aged exploit writers: If you were a ninja in the late 90s-early 00s, turn your attention to embedded devices, bootloaders and firmware. All your old skills are new again.

Traduire le Tweet

12:30 AM · 19 mai 2022 · Twitter for iPhone
Send arbitrary commands to the chip

- Linux ftrace framework
- No need to build a custom iwlmvm.ko
- Hijack a single function: iw1_mvm_send_cmd()
  - Custom requests from userland
  - Communicate through /sys/kernel/debug/iwlwifi/*/iwlmvm

```
$ make
make -C /lib/modules/4.15.0-177-generic/build M=/home/user/hook-driver
modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-177-generic'
  CC [M] /home/user/hook-driver/exploit.o
  CC [M] /home/user/hook-driver/ftrace_hook.o
  LD [M] /home/user/hook-driver/pwn.o
Building modules, stage 2.
MODPOST 1 modules
  CC /home/user/hook-driver/pwn.mod.o
  LD [M] /home/user/hook-driver/pwn.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-177-generic'
```
Exploit

- rwx region, no mitigations
- Put the shellcode in a global buffer thanks to a specific command
- Optional: read memory to ensure that the shellcode was successfully written
- Trigger the vulnerability
Payload – enable debug mode

```
$ sudo ./iwldebug.py read 0xc0887ff4 16
c0887ff4: efbe adde efbe adde efbe adde efbe adde

$ sudo ./iwldebug.py write 0xc0887ff4 61626364
Failed to write 4 bytes to 0xc0887ff4 (61626364)

$ sudo ./exploit_enable_debug.py
[*] loading module pwn
[*] putting shellcode in memory (24 bytes)
[*] ensuring shellcode is there
[*] triggering overflow
[*] ensuring debug flag is set
    SUCCESS (read at 0xc0a03088: 0x400)!
[*] unloading module pwn

$ sudo ./iwldebug.py write 0xc0887ff4 61626364
$ sudo ./iwldebug.py read 0xc0887ff4 16
 c0887ff4: 6162 6364 efbe adde efbe adde efbe adde
```
Loading patched firmware
Discovering the Loader

TOCTOU attack? (Transmit FW, Verify FW, Transmit patched FW)
Can Linux modify the data or the stack?

Linux

Wi-Fi chip

Memory
00060000-00061eff: loader
00401000-0040243b: data
00402e80-00402fff: stack

Registers
LMAC pc = 0x0006107e

Transmit FW
Get pc value

iwlwifi

SECURE

VULN
Bypassing the signature verification

The attack:

- Load modified firmware
- Overwrite a return address on the stack
- Profit $$$

🎉 SUCCESS on the first chip (Intel Wireless-AC 8260)

⛔ FAIL on the second one (Intel Wireless-AC 9560)

... probably because of the data cache

Make the chip commit its data cache ⇒ 🎉 SUCCESS on the second chip
Dynamic analysis
Tracing

- Tell which functions are executed
- Replace the first instruction (push_s blink) of every functions with:
  - LMAC: trap_s θ
  - UMAC: invalid instruction
- Hook the exception vector in the exception handler
  - Log the address to a unused buffer (0xc004ad00 - 0xc0050000)
  - Emulate push_s blink and return after the patched instruction
- Write hooks thanks to debug mode
- Read the shared buffer from the host in a loop
Working tracer

- Tracer injected using debugfs
- Unexpected difficulties
  - An integrity check mechanism prevents some UMAC pages from being modified
  - Patching some functions trigger a machine check (double fault?)
On-Chip Debugger

Goals: retrieve memory and register values to ease reverse engineering
On-Chip Debugger

- A debugger stub (PIC) is written to a fixed address
- 4 commands:
  - Read register
  - Write to memory (1 / 2 / 4 bytes)
  - Read from memory (1 / 2 / 4 bytes)
  - Resume execution
- Communication with the host through unused registers
- Targeted function pointers are replaced with the debugger address
- Allows to instrument a set of UMAC/LMAC functions
- Less powerful than a GDB stub
InVitroDbg

- Idea from Guillaume Delugré
  - Closer to metal: Reverse engineering the Broadcom NetExtreme's firmware
  - Hack.lu 2010
- Emulate firmware
  - Firmware execution on the host
  - Forward some memory accesses to the on-chip debugger
  - QEMU user with custom TCG plugin
  - GDB server
Firmware emulation with IO memory accesses
The Paging Memory

and how it uses DMA
The Additional Code in the File

Firmware File

- NUM_OF_CPU 2
- SEC_RT 00404000
- SEC_RT 00800000
- SEC_RT 00000000
- SEC_RT 00456000

- SEC_RT 00405000
- SEC_RT c0080000
- SEC_RT c0880000
- SEC_RT 80448000

Wi-Fi chip

Memory

- 00000000..00037fff (229376 bytes)
- c0080000..c008ffff (65536 bytes)
- 00404000..004042b7 (696 bytes)
- 00405000..004052b7 (696 bytes)
- 80448000..80455ad3 (56020 bytes)
- 00456000..0048d873 (227444 bytes)
- 00800000..00817fff (98304 bytes)
- c0880000..c0887fff (32768 bytes)
The Additional Code in the File

### Firmware File

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_OF_CPU</td>
<td>2</td>
</tr>
<tr>
<td>SEC_RT</td>
<td>00404000</td>
</tr>
<tr>
<td></td>
<td>00800000</td>
</tr>
<tr>
<td></td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td>00456000</td>
</tr>
<tr>
<td>SEC_RT</td>
<td>00405000</td>
</tr>
<tr>
<td></td>
<td>c0080000</td>
</tr>
<tr>
<td></td>
<td>c0880000</td>
</tr>
<tr>
<td></td>
<td>80448000</td>
</tr>
<tr>
<td>SEC_RT</td>
<td>aaaabbbb</td>
</tr>
<tr>
<td></td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td>01000000</td>
</tr>
</tbody>
</table>

### Wi-Fi chip

#### Memory

<table>
<thead>
<tr>
<th>Memory Segment</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000..00037fff</td>
<td>229376</td>
</tr>
<tr>
<td>c0080000..c008ffff</td>
<td>65536</td>
</tr>
<tr>
<td>00404000..004042b7</td>
<td>696</td>
</tr>
<tr>
<td>00405000..004052b7</td>
<td>696</td>
</tr>
<tr>
<td>80448000..80455ad3</td>
<td>56020</td>
</tr>
<tr>
<td>00456000..0048d873</td>
<td>227444</td>
</tr>
<tr>
<td>00800000..00817fff</td>
<td>98304</td>
</tr>
<tr>
<td>c0880000..c0887fff</td>
<td>32768</td>
</tr>
<tr>
<td>aaaabbbb: separator (4 bytes)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>00000000..00000297</td>
<td>664</td>
</tr>
<tr>
<td>01000000..0103aff</td>
<td>241664</td>
</tr>
</tbody>
</table>
Memory Management Unit (MMU)

The Paging Memory is like Linux's swap mechanism

UMAC virtual memory

UMAC physical memory

Main physical memory (managed by Linux)

“Paging Memory”
Data stored by iwlwifi
(236 KiB, 59 pages)

Other memory

00000000-3fffffff

DMA request if needed

DMA Attack

MMU Configuration

01000xxx is not present
01001xxx is not present
01002xxx is at 00432xxx
...

UMAC physical memory

00422000-00447fff
(152 KiB, 38 pages)

00000000-3fffffff

UMAC virtual memory

01000000-01fffffff
80000000-bfffffff
c0000000-xffffffff
Demo!

https://asciinema.org/a/yYm5aKRdnO16TfJtCjKZVhGV0
(Ab)using The Paging Memory

The host physical addresses are used/managed by the chip. Can it do arbitrary DMA requests?

- YES! Demo!

What about the IOMMU?

- By default on Ubuntu, the IOMMU is not enabled
- Protection: add `intel_iommu=on` to the kernel command line
Conclusion
TDLS crash analysis

- Not exploitable
- Update not available on some Linux distros (eg. Ubuntu 18.04 LTS)
- Remote firmware crash with a single Wi-Fi packet
Conclusion

- Many things left to explore:
  - Bluetooth interface
  - WoWLAN (Wake-on-Wireless Local Area Network)
  - ThreadX usage by the UMAC code
  - Wi-Fi frame parsing and vulnerability research
- Open-source tools: [https://github.com/Ledger-Donjon/intel-wifi-research-tools](https://github.com/Ledger-Donjon/intel-wifi-research-tools)
- Groundwork for other security researchers
Questions?