LEDGER

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

Context

```
# 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: 0x00000070 | 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

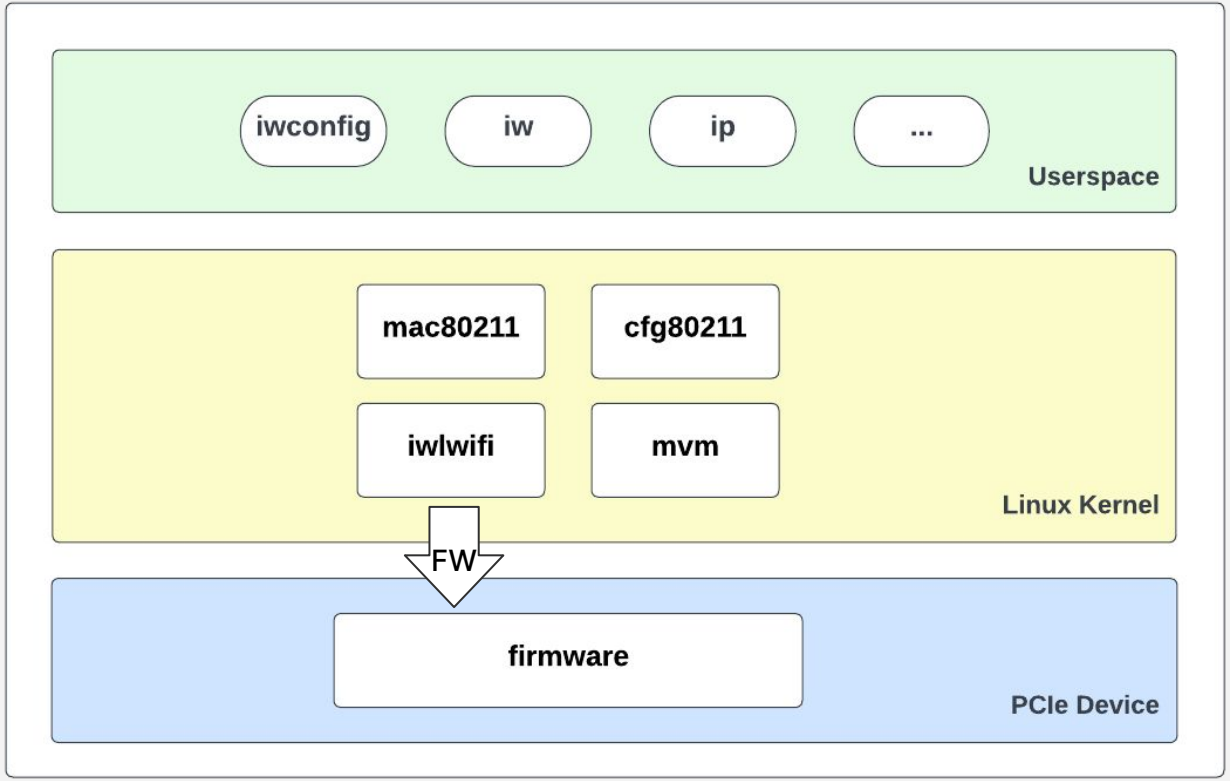
Intel Wireless-AC 9560
(Picture of a Companion RF Module)

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)

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

```
# 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

https://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git/

Firmware file format

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

Entries:
- Type,
- Length,
- Value

```
00000000: 0000 0000 4957 4c0a 7265 6c65 6173 652f  ....IWL.release/
00000010: 636f 7265 3433 3a3a 3666 3966 3231 3563  core43::6f9f215c
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 2e00 0000 5c21 9f6f  ............\!.o
00000050: 0000 0000 0000 0000 1600 0000 0c00 0000  ................
00000060: 0000 0000 db15 060f 8b95 020f 2400 0000  ............$...
00000070: 0c00 0000 2e00 0000 5c21 9f6f 0000 0000  ........\!.o....
00000080: 3700 0000 2000 0000 143c 8100 7c74 4600  7... ....<..|tF.
…
000002e0: 0700 0000 0000 0000 1b00 0000 0400 0000  ................
000002f0: 0200 0000 1300 0000 bc02 0000 0040 4000  .............@@.
00000300: 0600 0000 a100 0000 0000 0100 0000 0000  ................
00000310: 8680 0000 2801 2120 cb1e 0200 4000 0000  ....(.! ....@...
```

No encryption

Firmware file format

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

```
IWL_UCODE_TLV_FLAGS             = 18,
IWL_UCODE_TLV_SEC_RT            = 19,
IWL_UCODE_TLV_SEC_INIT          = 20,
IWL_UCODE_TLV_SEC_WOWLAN        = 21,
IWL_UCODE_TLV_DEF_CALIB         = 22,
IWL_UCODE_TLV_PHY_SKU           = 23,
IWL_UCODE_TLV_SECURE_SEC_RT     = 24,
IWL_UCODE_TLV_SECURE_SEC_INIT   = 25,
IWL_UCODE_TLV_SECURE_SEC_WOWLAN = 26,
IWL_UCODE_TLV_NUM_OF_CPU        = 27,
IWL_UCODE_TLV_CSCHEME           = 28,
IWL_UCODE_TLV_API_CHANGES_SET   = 29,
IWL_UCODE_TLV_ENABLED_CAPABILITIES      = 30,
IWL_UCODE_TLV_N_SCAN_CHANNELS           = 31,
IWL_UCODE_TLV_PAGING            = 32,
```

```
0000 0000 4957 4c0a 7265 6c65 6173 652f  ....IWL.release/
636f 7265 3433 3a3a 3666 3966 3231 3563  core43::6f9f215c
0000 0000 0000 0000 0000 0000 0000 0000  ................
0000 0000 0000 0000 0000 0000 0000 0000  ................
0000 0000 0000 0000 2e00 0000 5c21 9f6f  ............\!.o
0000 0000 0000 0000 1600 0000 0c00 0000  ................
0000 0000 db15 060f 8b95 020f 2400 0000  ............$...
0c00 0000 2e00 0000 5c21 9f6f 0000 0000  ........\!.o....
3700 0000 2000 0000 143c 8100 7c74 4600  7... ....<..|tF.

0700 0000 0000 0000 1b00 0000 0400 0000  ................
0200 0000 1300 0000 bc02 0000 0040 4000  .............@@.
00000300: 0600 0000 a100 0000 0000 0100 0000 0000  ................
00000310: 8680 0000 2801 2120 cb1e 0200 4000 0000  ....(.! ....@...
```

No encryption

Firmware decoder

```
$ 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)
```

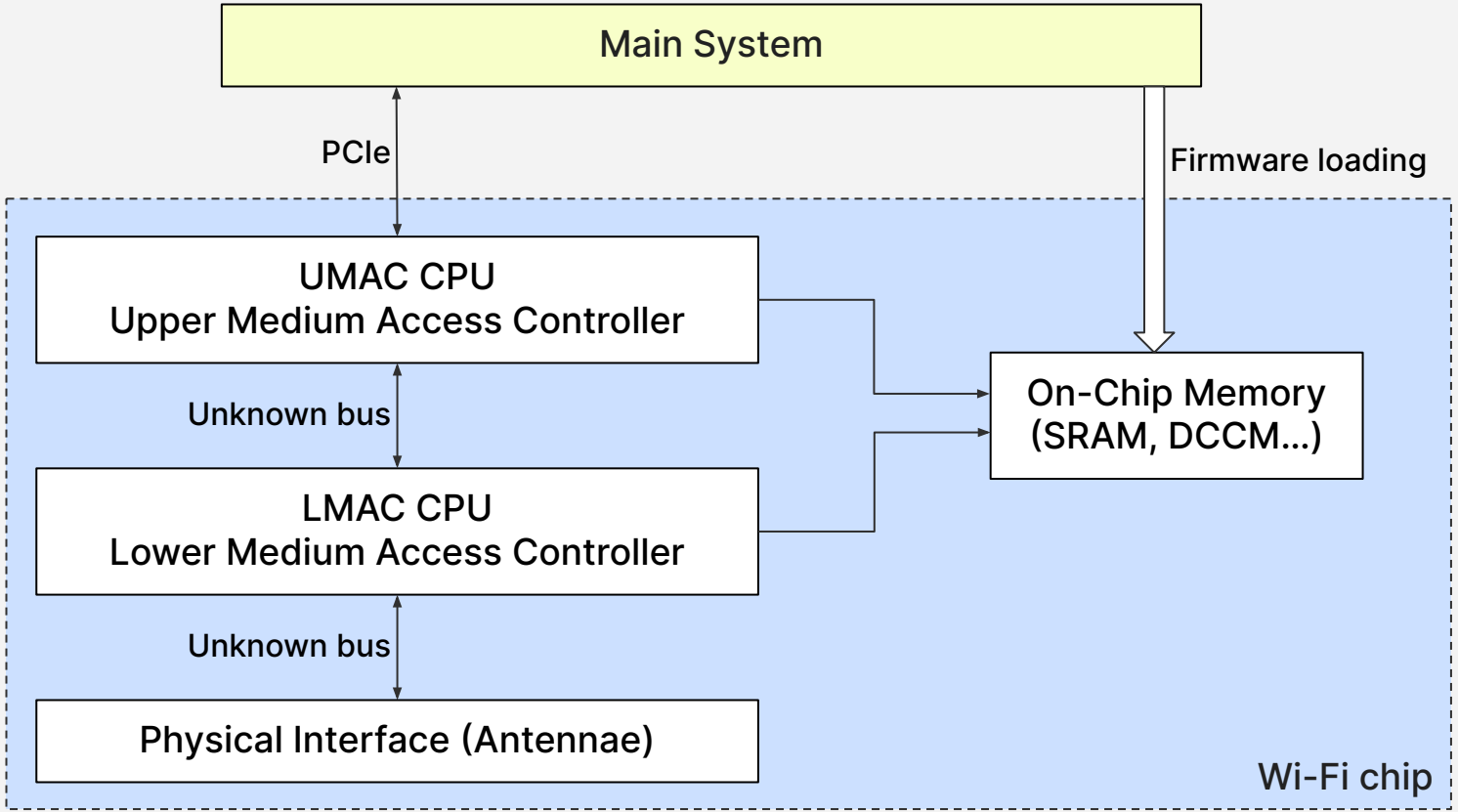
```
IWL_UCODE_TLV_SEC_RT          = 19,
IWL_UCODE_TLV_SEC_INIT        = 20,
IWL_UCODE_TLV_SEC_WOWLAN      = 21,
IWL_UCODE_TLV_DEF_CALIB       = 22,
IWL_UCODE_TLV_PHY_SKU         = 23,
IWL_UCODE_TLV_SECURE_SEC_RT   = 24,
IWL_UCODE_TLV_SECURE_SEC_INIT = 25,
IWL_UCODE_TLV_SECURE_SEC_WOWLAN = 26,
IWL_UCODE_TLV_NUM_OF_CPU      = 27,
IWL_UCODE_TLV_CSCHEME         = 28,
IWL_UCODE_TLV_API_CHANGES_SET = 29,
IWL_UCODE_TLV_ENABLED_CAPABILITIES      = 30,
IWL_UCODE_TLV_N_SCAN_CHANNELS           = 31,
IWL_UCODE_TLV_PAGING          = 32,
```

```
0000 0000 0000 0000 1600 0000 0c00 0000  ................
0000 0000 db15 060f 8b95 020f 2400 0000  ............$...
0c00 0000 2e00 0000 5c21 9f6f 0000 0000  ........\!.o....
3700 0000 2000 0000 143c 8100 7c74 4600  7... ....<..|tF.

0700 0000 0000 0000 1b00 0000 0400 0000  ................
0200 0000 1300 0000 bc02 0000 0040 4000  .............@@.
0600 0000 a100 0000 0000 0100 0000 0000  ................
8680 0000 2801 2120 cb1e 0200 4000 0000  ....(.! ....@...
```

No encryption

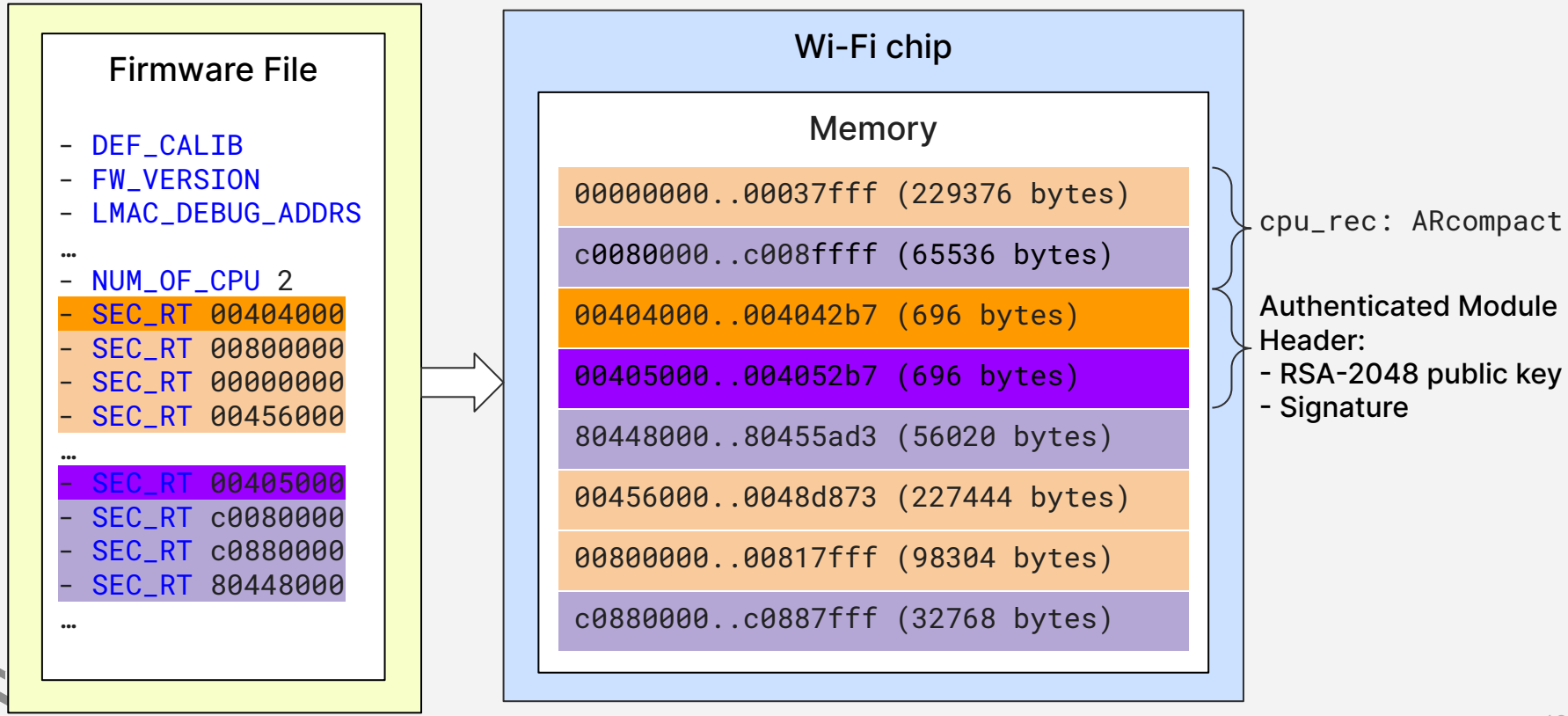
2 Processors?!?



Firmware memory layout

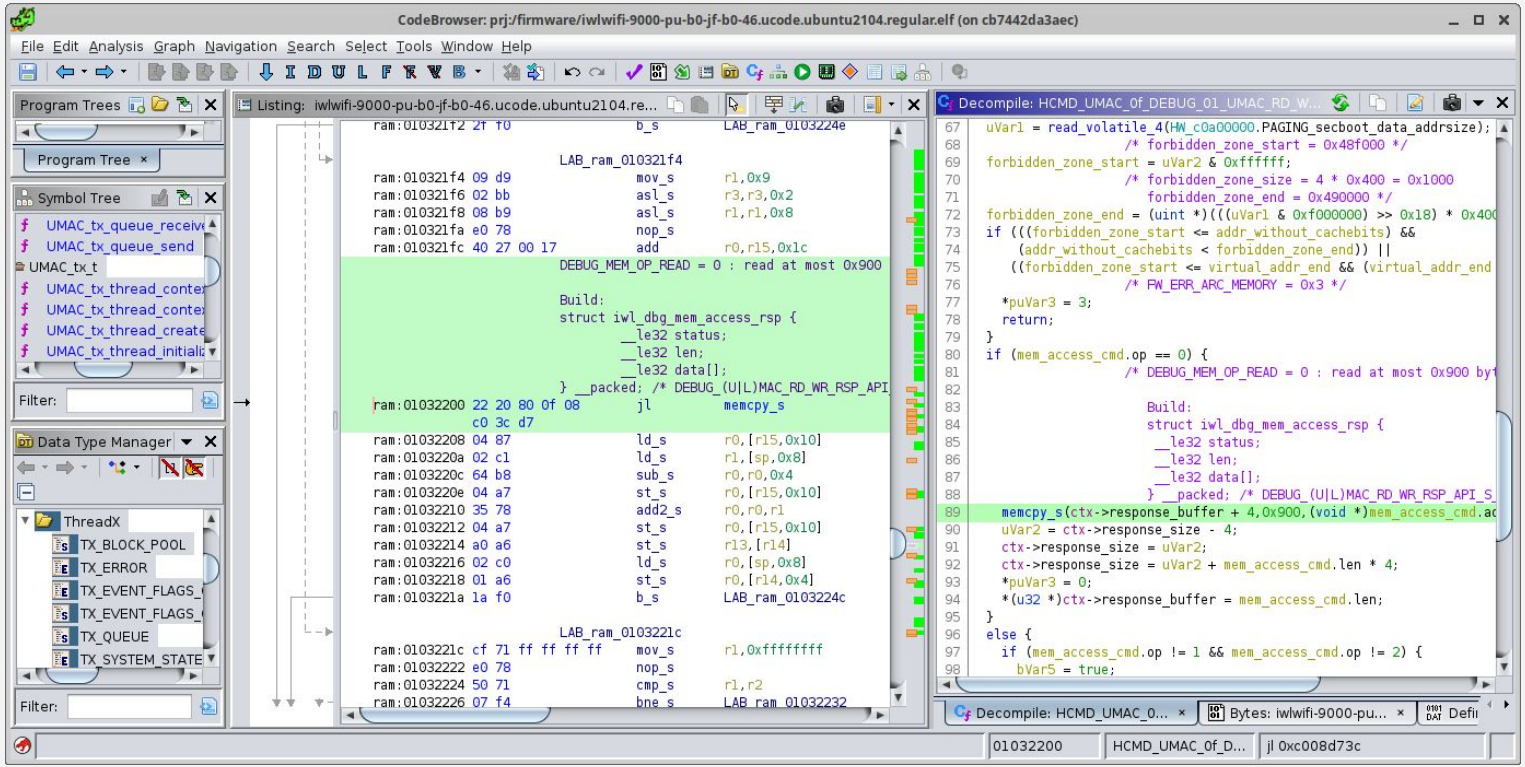
Linux: net/wireless/intel/iwlwifi/iwl-drv.c

```
#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

Talking to the Chip

Linux Debug Filesystem

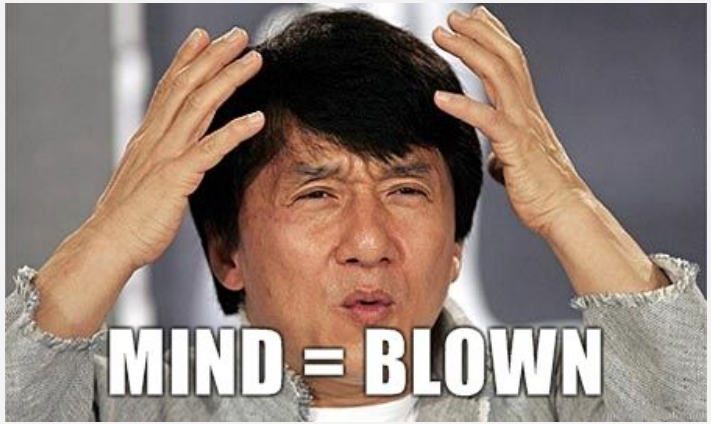
Memory read: almost anywhere :) (not `0048f000...0048ffff`)

```
# DBGFS=/sys/kernel/debug/iwlwifi/0000:00:14.3
# cat $DBGFS/iwlmvm/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=$DBGFS/iwlmvm/mem bs=1 count=128 | xxd
00000000: 2020 800f 0000 4000 2020 800f 0300 e474    ....@.  .....t
00000010: 2020 800f 0300 3837 2020 800f 0000 c819    ....87  ......
00000020: 6920 0000 6920 4000 6920 0000 6920 4000  i ..i @.i ..i @.
00000030: 2020 800f 4700 14b6 6920 0000 6920 4000    ..G...i ..i @.
00000040: 6920 0000 4a20 0000 4a21 0000 4a22 0000  i ..J ..J!..J"..
00000050: 4a23 0000 4a24 0000 4a25 0000 4a26 0000  J#..J$..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&..
```

Getting the PC (Program Counter)

```
// 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

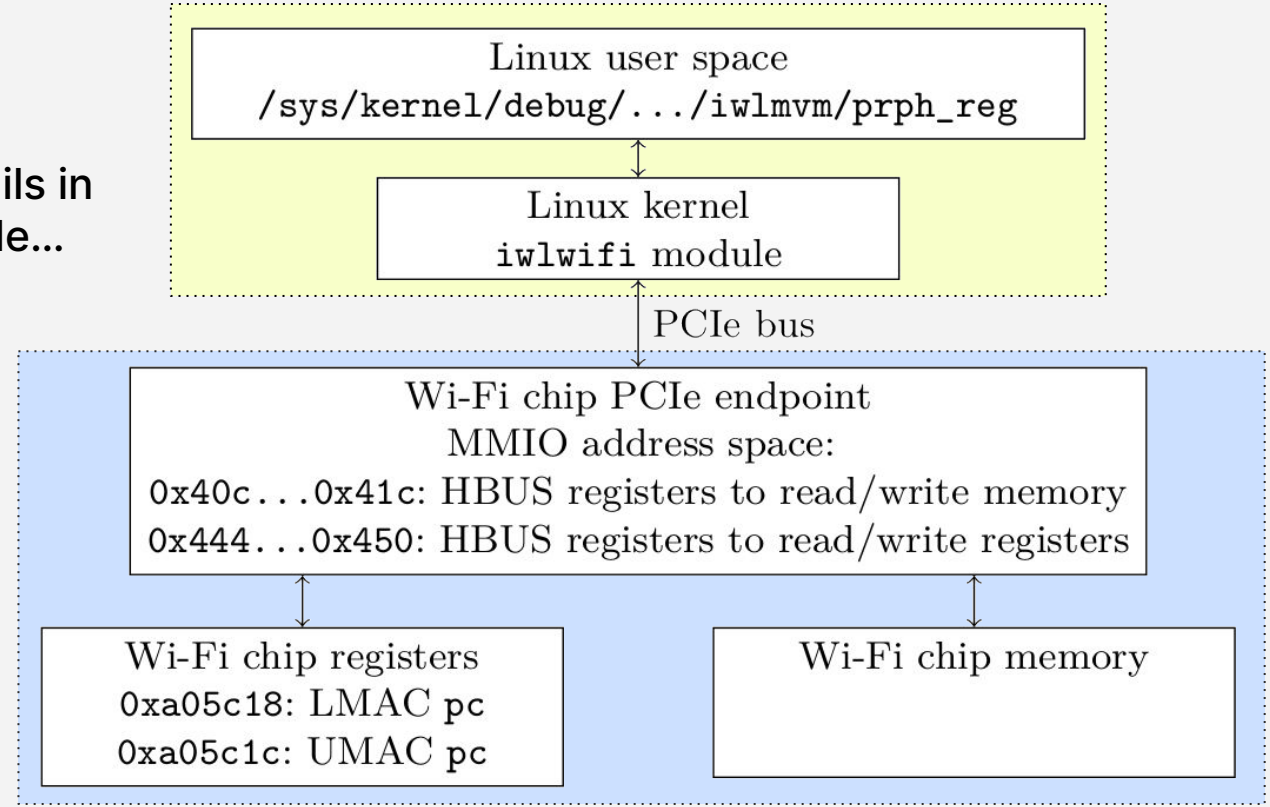
LMAC pc

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

```
enum iwl_mvm_command_groups {
        LEGACY_GROUP = 0x0,
        LONG_GROUP = 0x1,
        SYSTEM_GROUP = 0x2,
        MAC_CONF_GROUP = 0x3,
        PHY_OPS_GROUP = 0x4,
        DATA_PATH_GROUP = 0x5,
        NAN_GROUP = 0x7,
        LOCATION_GROUP = 0x8,
        PROT_OFFLOAD_GROUP = 0xb,
        REGULATORY_AND_NVM_GROUP = 0xc,
        DEBUG_GROUP = 0xf,
};
```

```
enum iwl_legacy_cmds {
        /**
         * @UCODE_ALIVE_NTFY:
         * Alive data from the firmware, as described in
         * &struct iwl_alive_ntf_v3 or &struct iwl_alive_ntf_v4 or
         * &struct iwl_alive_ntf_v5.
         */
        UCODE_ALIVE_NTFY = 0x1,

        /**
         * @REPLY_ERROR: Cause an error in the firmware, for testing purposes.
         */
        REPLY_ERROR = 0x2,

        /**
         * @ECHO_CMD: Send data to the device to have it returned immediately.
         */
        ECHO_CMD = 0x3,
```

Vulnerability

```
CODE:C0087A58                           sub_C0087A58:                           # DATA XREF: data:LEGACY_GROUP↓o
CODE:C0087A58
CODE:C0087A58                           size            = -0x6C
CODE:C0087A58                           flag            = -0x68
CODE:C0087A58                           buffer          = -0x64
CODE:C0087A58                           var_14          = -0x14
CODE:C0087A58                           var_10          = -0x10
CODE:C0087A58
CODE:C0087A58 E1 C5                                     push    r13
CODE:C0087A5A E1 C6                                     push    r14
CODE:C0087A5C F1 C0                                     push    blink
CODE:C0087A5E B8 C1                                     sub     sp, sp, 0x60 # '`'
CODE:C0087A60 08 75                                     mov     r13, r0
CODE:C0087A62 8B 70                                     mov     r0, sp          # buffer
CODE:C0087A64 02 D9                                     mov     r1, 2           # count
CODE:C0087A66 00 DE                                     mov     r14, 0
CODE:C0087A68 C9 72                                     mov     r2, r14
CODE:C0087A6A 22 20 80 0F 08 C0 60 E1     jl      umac_fifo_read_bytes # read 2 dwords
CODE:C0087A72 00 C1                                     ld      r1, [sp,0x6C+size] # count
CODE:C0087A74 80 E1                                     cmp     r1, 0
CODE:C0087A76 08 F2                                     beq     loc_C0087A84
CODE:C0087A78 82 C0                                     add     r0, sp, 0x6C+buffer # buffer
CODE:C0087A7A C9 72                                     mov     r2, r14
CODE:C0087A7C 22 20 80 0F 08 C0 60 E1     jl      umac_fifo_read_bytes # read `count` dwords
```

Exploitation

Send arbitrary commands to the chip

- Linux ftrace framework
- No need to build a custom `iwlmvm.ko`
- Hijack a single function: `iwl_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

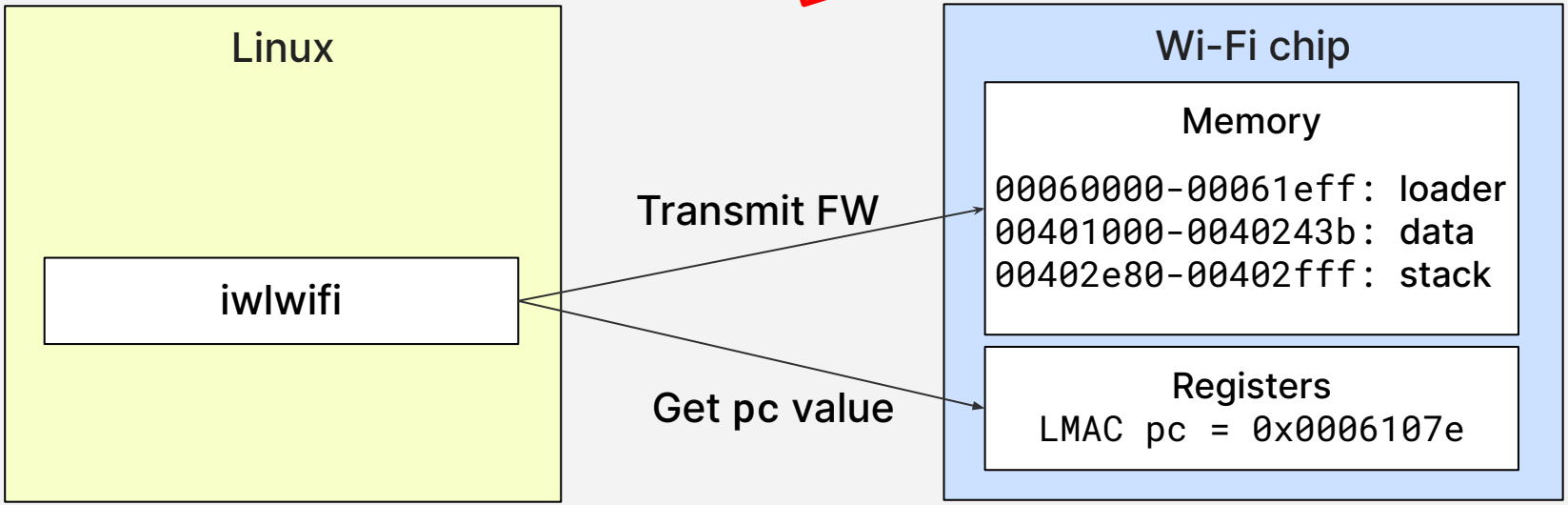
Discovering the Loader

TOCTOU attack? (Transmit FW, Verify FW, Transmit patched FW)

Can Linux modify the data or the stack?

VULN

SECURE



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

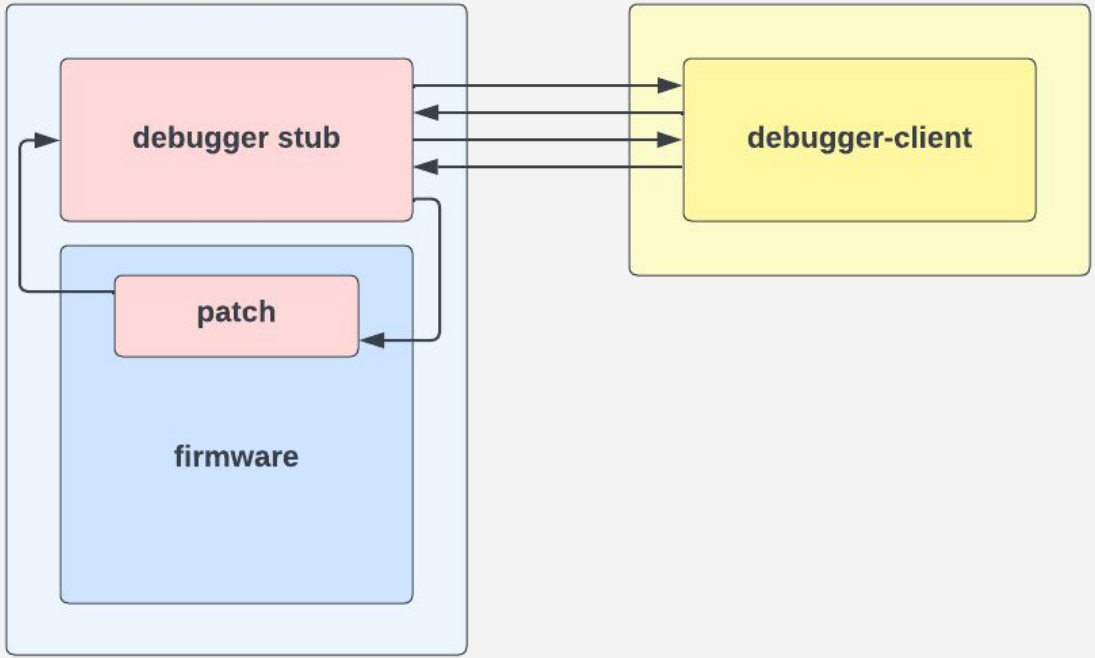
- Tell which functions are executed
- Replace the first instruction (`push_s blink`) of every functions with:
  - LMAC: `trap_s 0`
  - UMAC: invalid instruction
- Hook the exception vector in the exception handler
  - Log the address to a unused buffer (0xc004ad00 - 0xc0050000)
  - Emulate `push_s blink` 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



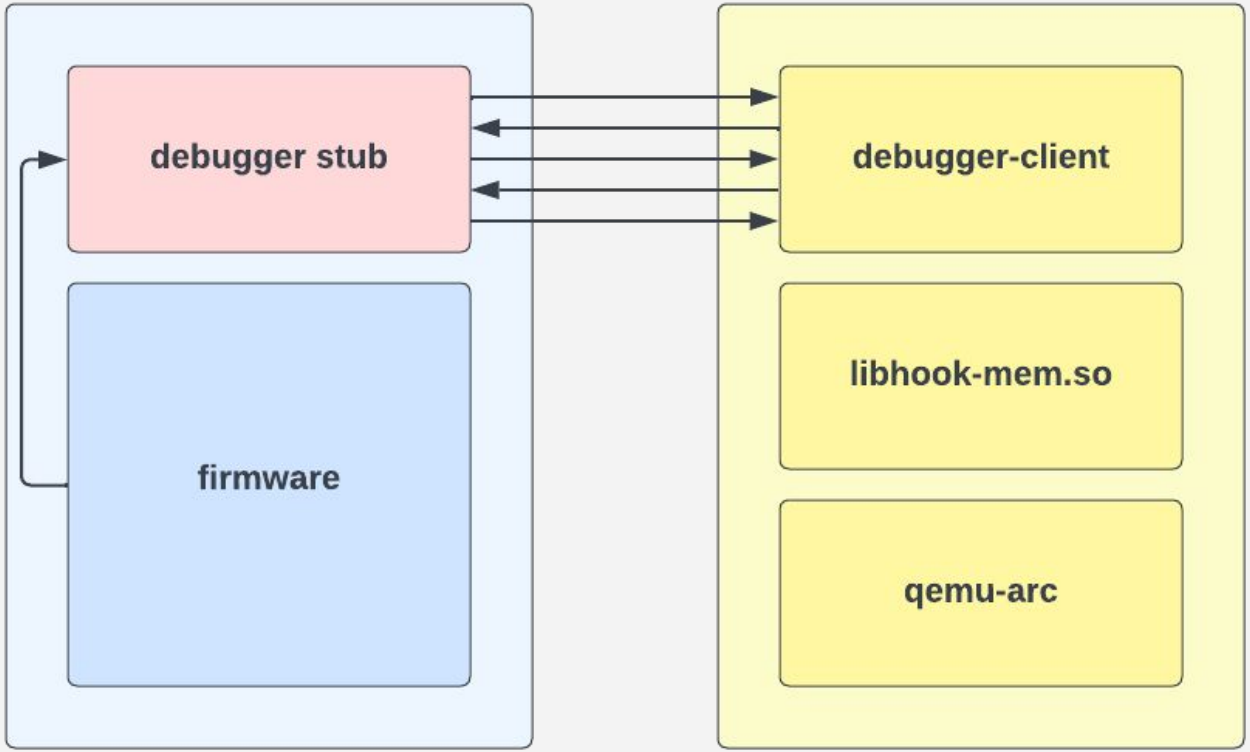
On-Chip Debugger

- 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

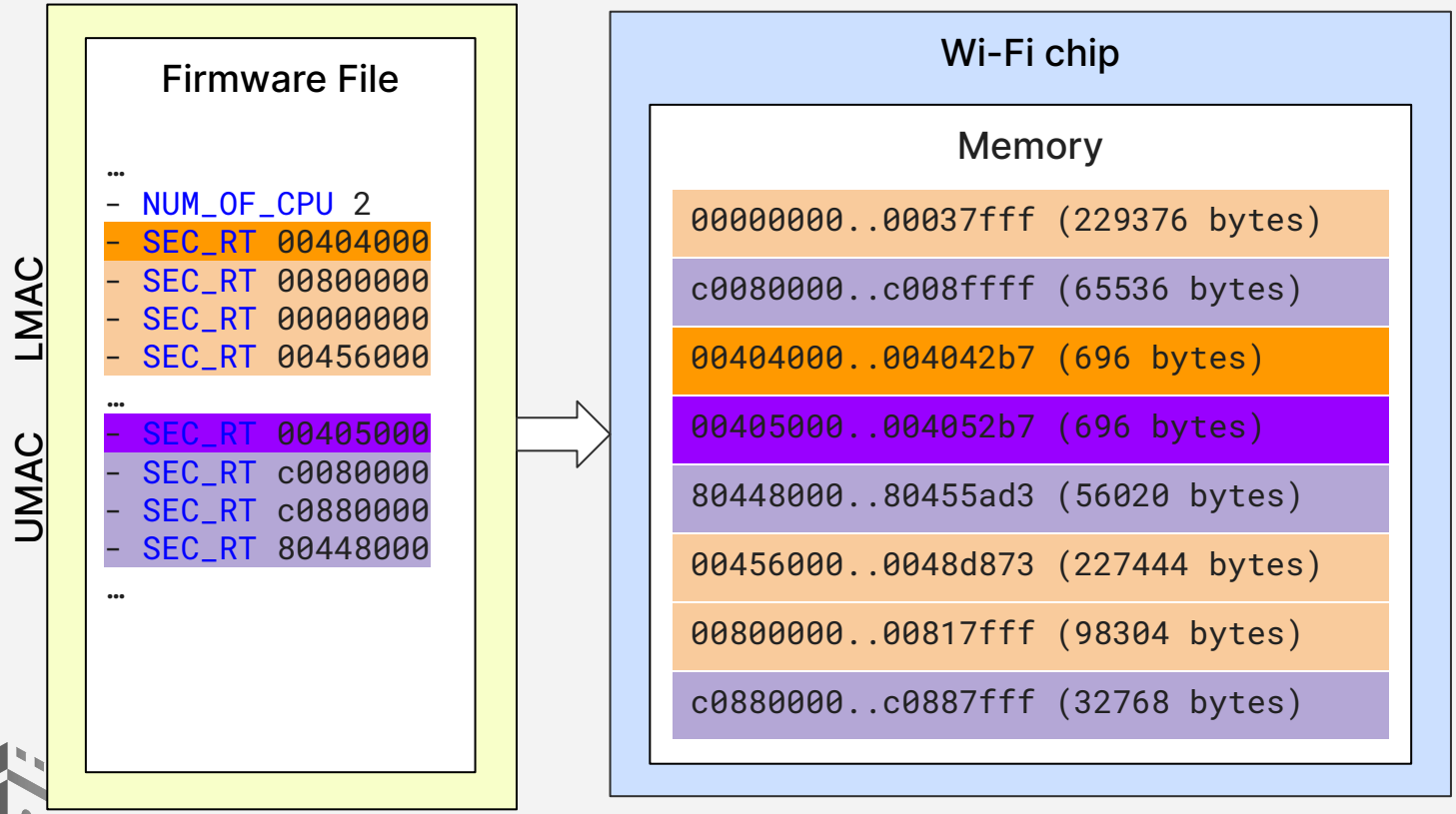
Firmware emulation with IO memory accesses



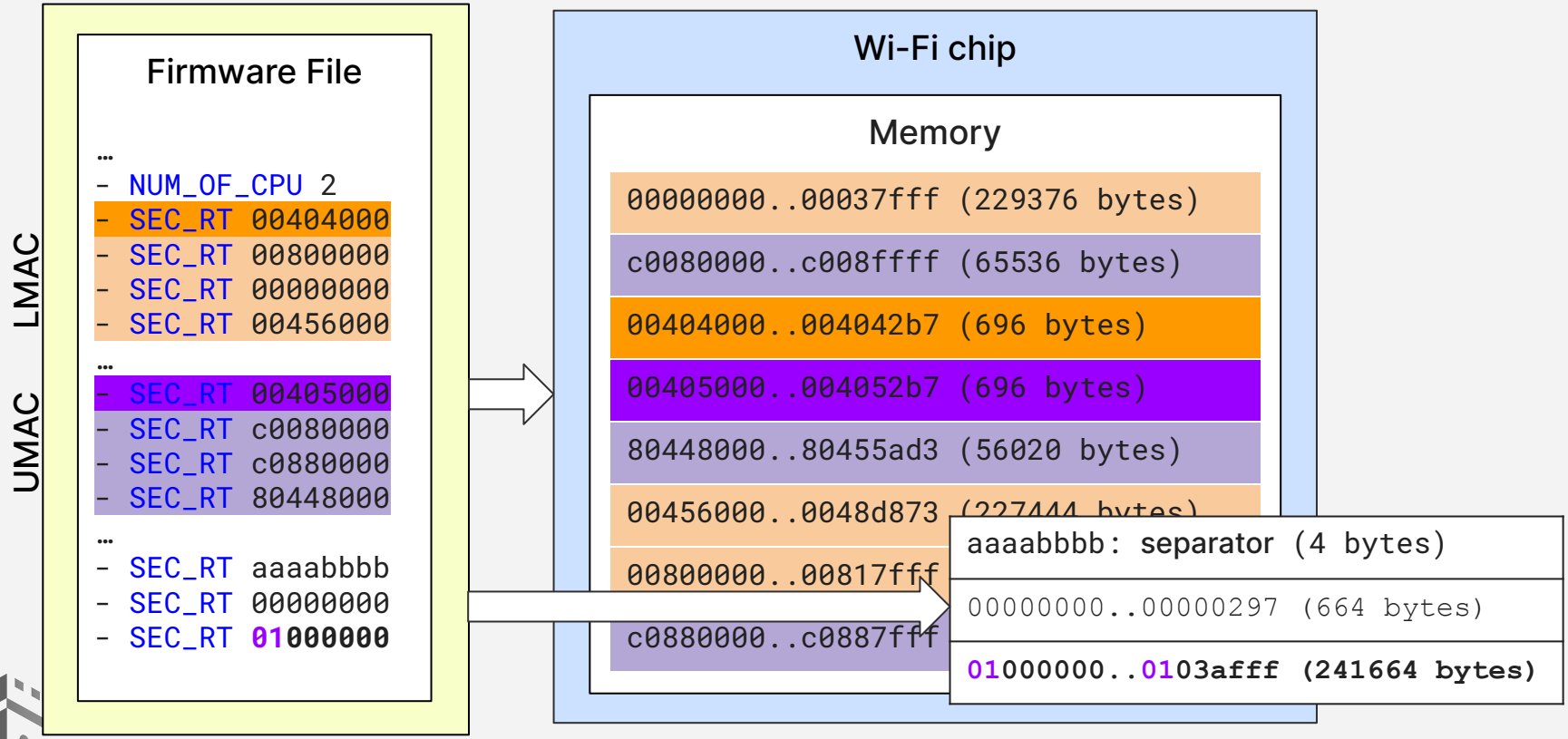
The Paging Memory

and how it uses DMA

The Additional Code in the File

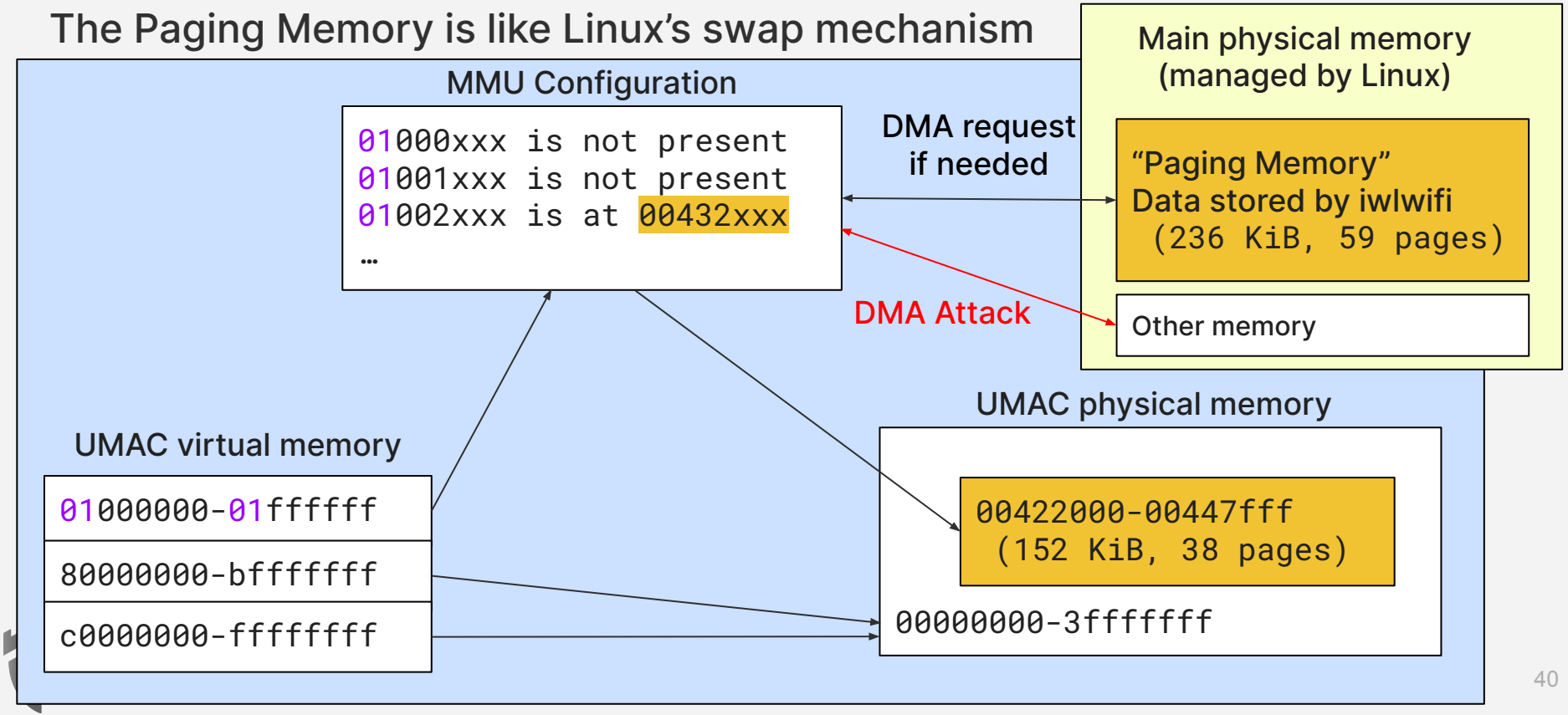


The Additional Code in the File



Memory Management Unit (MMU)

The Paging Memory is like Linux's swap mechanism



Demo!

https://asciinema.org/a/yYm5aKRdnO16TfJTcjKZVhGV0

```
wifi@test $ ./dbg_show_paging_ucode_info.py | grep '0x01040000 '
  (ff) 0x01040000 -> -                (host 0x256ba0000)
wifi@test $ ./iwldebug.py read 0xc0885774 64
c0885774: 0103 0000 0f00 0000 0900 0000 ea2a 2500  .............*%.
c0885784: f852 3c00 3878 3200 5899 3800 d82a 1100  .R<.8x2.X.8..*..
c0885794: 1841 1100 5021 3a00 c06b 2500 1859 3500  .A..P!:..k%..Y5.
c08857a4: a06b 2500 0000 0000 0000 0000 0000 0000  .k%.............
wifi@test $ ./iwldebug.py write 0xc08857a4 00862300
wifi@test $ ./dbg_show_paging_ucode_info.py | grep '0x01040000 '
  (ff) 0x01040000 -> -                (host 0x238600000)
wifi@test $ ./iwldebug.py read 0x01040100 256
01040100: 616c 6c73 0000 0000 0000 0000 0000 0000  alls............
01040110: 7365 7475 705f 636f 6d6d 616e 645f 6c69  setup_command_li
01040120: 6e65 0000 0000 0000 7374 726e 6c65 6e00  ne......strnlen.
01040130: 7374 726c 656e 0000 696e 6974 6361 6c6c  strlen..initcall
01040140: 5f64 6562 7567 0000 696e 6974 6361 6c6c  _debug..initcall
01040150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
01040160: 2573 2076 6572 7369 6f6e 2025 7320 2862  %s version %s (b
01040170: 7569 6c64 6440 6c63 7930 322d 616d 6436  uildd@lcy02-amd6
01040180: 342d 3031 3629 2028 6763 6320 2855 6275  4-016) (gcc (Ubu
01040190: 6e74 7520 3131 2e32 2e30 2d37 7562 756e  ntu 11.2.0-7ubun
010401a0: 7475 3229 2031 312e 322e 302c 2047 4e55  tu2) 11.2.0, GNU
010401b0: 206c 6420 2847 4e55 2042 696e 7574 696c   ld (GNU Binutil
010401c0: 7320 666f 7220 5562 756e 7475 2920 322e  s for Ubuntu) 2.
010401d0: 3337 2920 2573 0a00 0000 0000 0000 0000  37) %s..........
010401e0: 4c69 6e75 7820 7665 7273 696f 6e20 352e  Linux version 5.
010401f0: 3133 2e30 2d34 342d 6765 6e65 7269 6320  13.0-44-generic
wifi@test $
```

```
00d0  0000 0000 0000 0000 0000 0000 0000 0000  ................
00e0  0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0  0000 0000 0000 0000 0000 0000 0000 0000  ................
root@test # grep Kernel /proc/iomem
  237400000-238402506 : Kernel code
  238600000-239045fff : Kernel rodata
  239200000-23956dfbf : Kernel data
  239867000-239dfffff : Kernel bss
root@test # python-chipsec read 0x238600100 256
****** Chipsec Linux Kernel module is licensed under GPL 2.0
[CHIPSEC] API mode: using CHIPSEC kernel module API
0000  616C 6C73 0000 0000 0000 0000 0000 0000  alls............
0010  7365 7475 705F 636F 6D6D 616E 645F 6C69  setup_command_li
0020  6E65 0000 0000 0000 7374 726E 6C65 6E00  ne......strnlen.
0030  7374 726C 656E 0000 696E 6974 6361 6C6C  strlen..initcall
0040  5F64 6562 7567 0000 696E 6974 6361 6C6C  _debug..initcall
0050  0000 0000 0000 0000 0000 0000 0000 0000  ................
0060  2573 2076 6572 7369 6F6E 2025 7320 2862  %s version %s (b
0070  7569 6C64 6440 6C63 7930 322D 616D 6436  uildd@lcy02-amd6
0080  342D 3031 3629 2028 6763 6320 2855 6275  4-016) (gcc (Ubu
0090  6E74 7520 3131 2E32 2E30 2D37 7562 756E  ntu 11.2.0-7ubun
00a0  7475 3229 2031 312E 322E 302C 2047 4E55  tu2) 11.2.0, GNU
00b0  206C 6420 2847 4E55 2042 696E 7574 696C   ld (GNU Binutil
00c0  7320 666F 7220 5562 756E 7475 2920 322E  s for Ubuntu) 2.
00d0  3337 2920 2573 0A00 0000 0000 0000 0000  37) %s..........
00e0  4C69 6E75 7820 7665 7273 696F 6E20 352E  Linux version 5.
00f0  3133 2E30 2D34 342D 6765 6E65 7269 6320  13.0-44-generic
root@test #
```

(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?

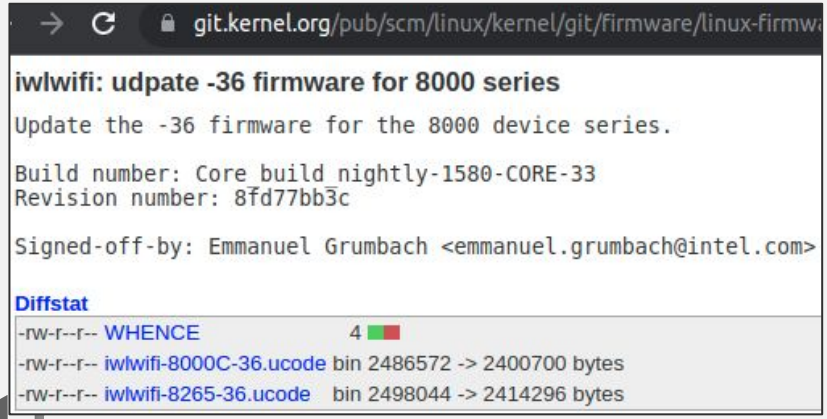
- By default on Ubuntu, the IOMMU is not enabled
- Protection: add `intel_iommu=on` to the kernel command line

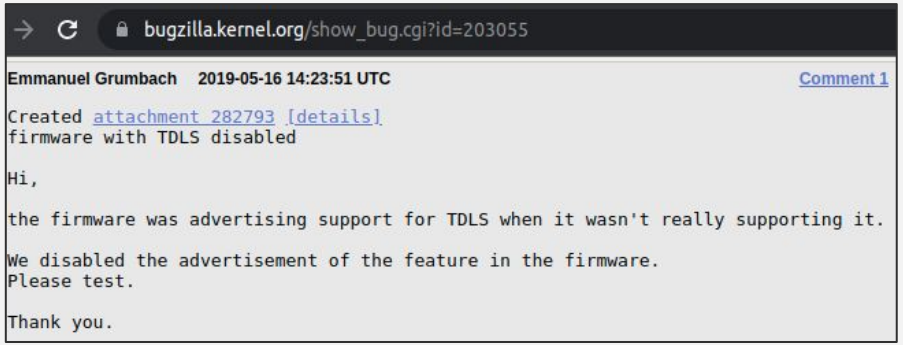
```
[  259.578089] DMAR: DRHD: handling fault status reg 3
[  259.578094] DMAR: [DMA Read] Request device [00:14.3] PASID ffffffff fault
addr 406a00000 [fault reason 06] PTE Read access is not set
[  261.600645] iwlwifi 0000:00:14.3: Error sending UNKNOWN: time out after
2000ms.
…
[  261.601783] iwlwifi 0000:00:14.3: 0x00000084 | NMI_INTERRUPT_UNKNOWN
```

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
- Groundwork for other security researchers

Questions?