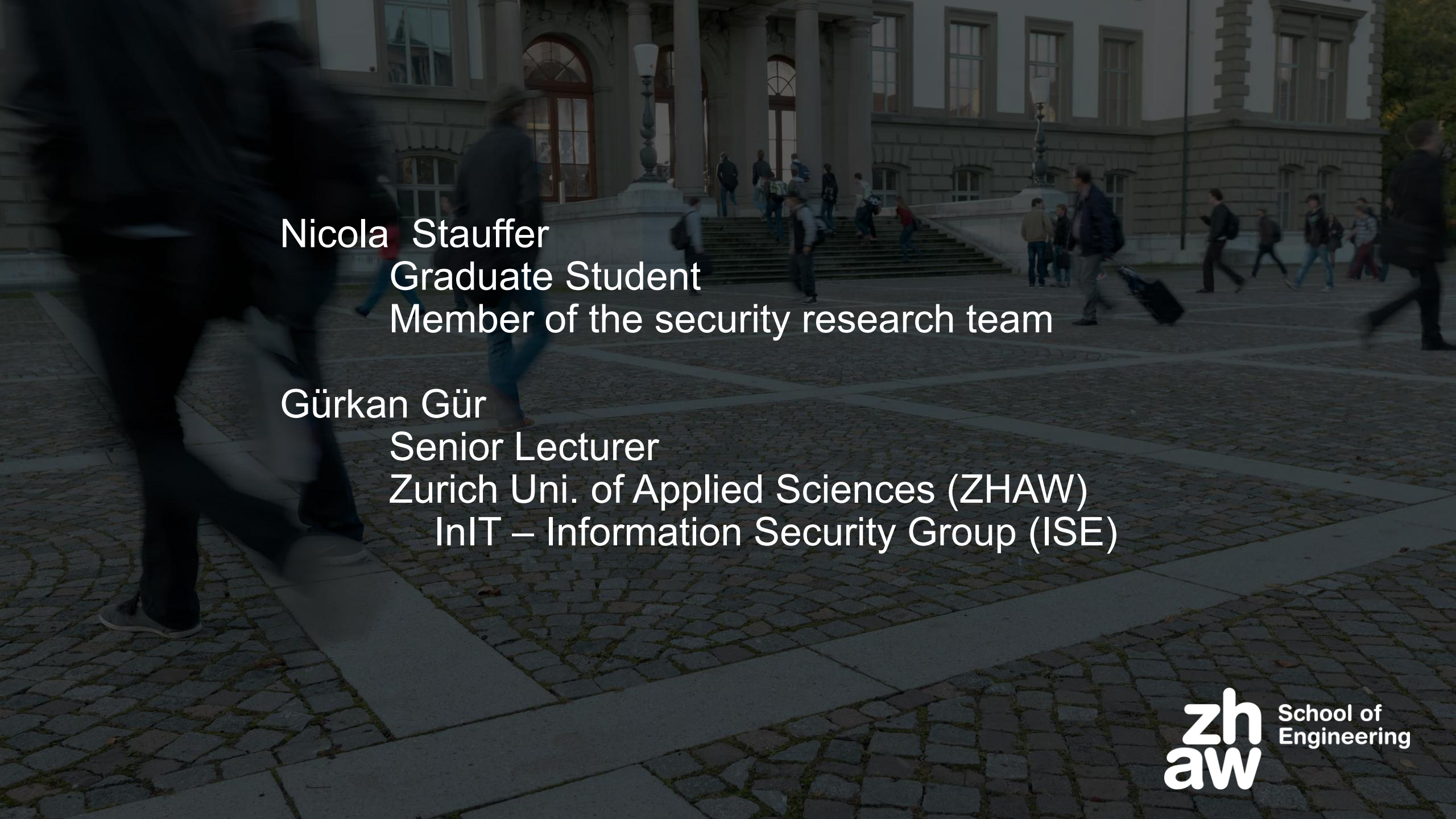
Overkill: Hijacking a Wi-Fi 7 Chip for SYSTEM Privileges

Nicola Stauffer | Gürkan Gür



Nicola Stauffer
Graduate Student
Member of the security research team

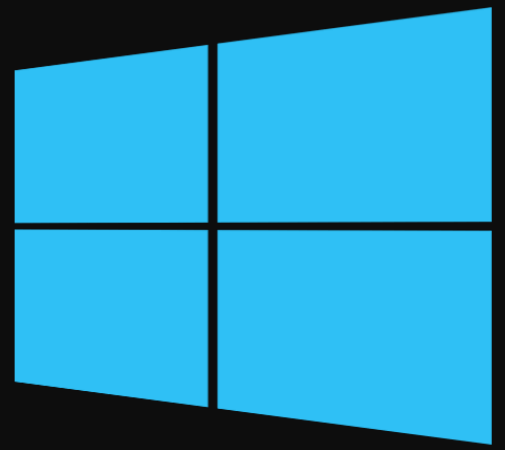
Gürkan Gür
Senior Lecturer
Zurich Uni. of Applied Sciences (ZHAW)
InIT – Information Security Group (ISE)

zh aw School of Engineering

Firmware







CVE-2015-6639 – TrustZone
CVE-2017-9417 – BroadPwn
CVE-2024-43047 – Qualcomm DSP

???


WHO'S THAT
POKÉMON?

Target

MediaTek, I choose you!



https://www.mediatek.com/products/broadband-wifi/mediatek-filogic-360

Target

- MT360 - Mass production started in 2024
- Single-chip Wi-Fi 7 2x2 and dual Bluetooth 5.4 combo
- 490 million to 650 million standalone Wi-Fi units in 2025

- And we had one laying around ☺

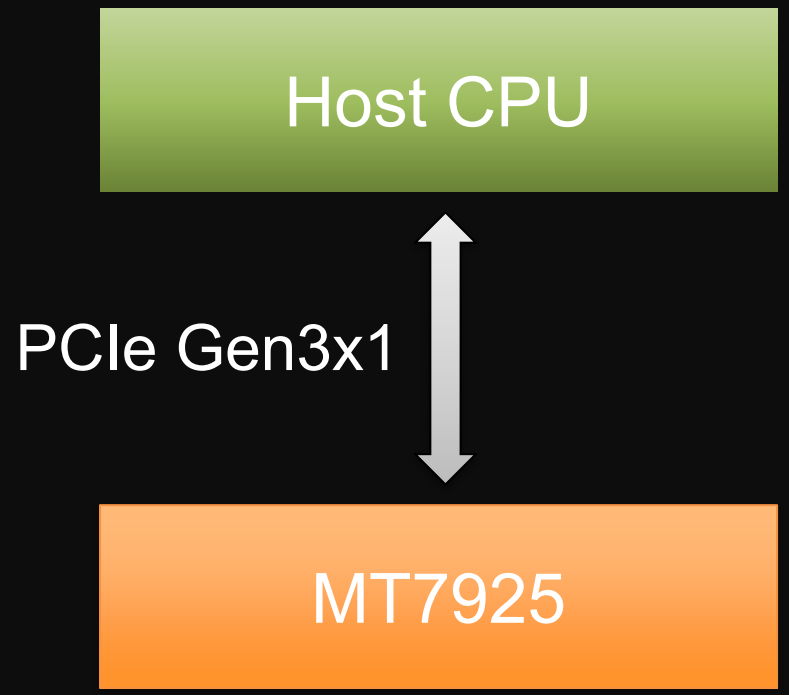
https://www.mediatek.com/products/broadband-wifi/mediatek-filogic-360



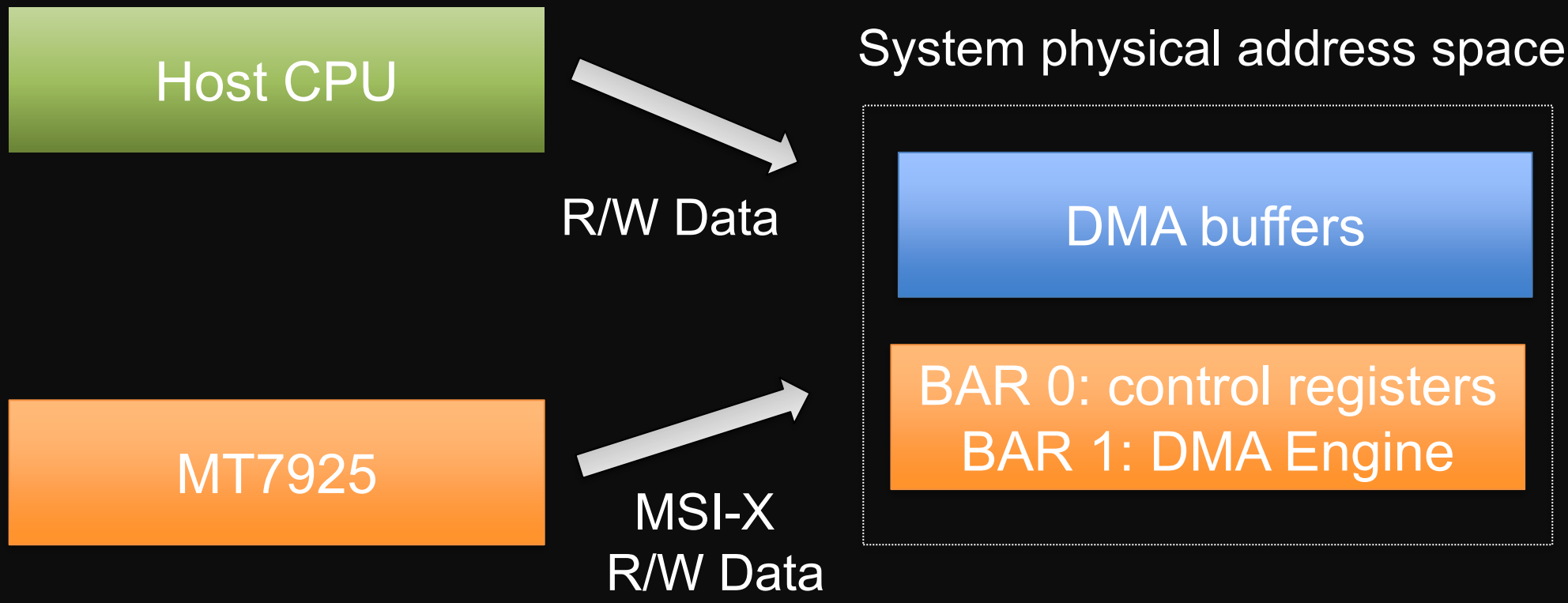
Basics

User mode Kernel mode Firmware

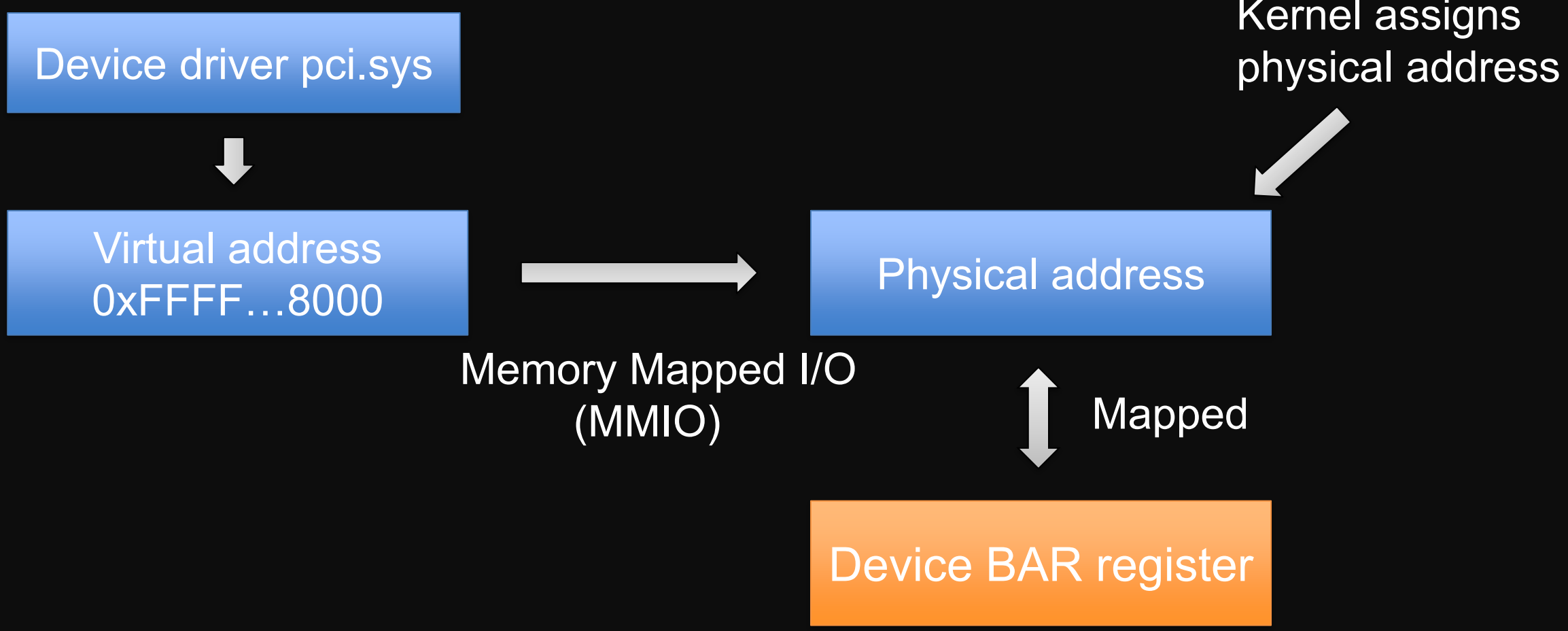
A peek inside your PC



A peek inside your PC



PCIe speedrun



Kernel - Firmware interaction

Mental health indicator



User mode Kernel mode Firmware

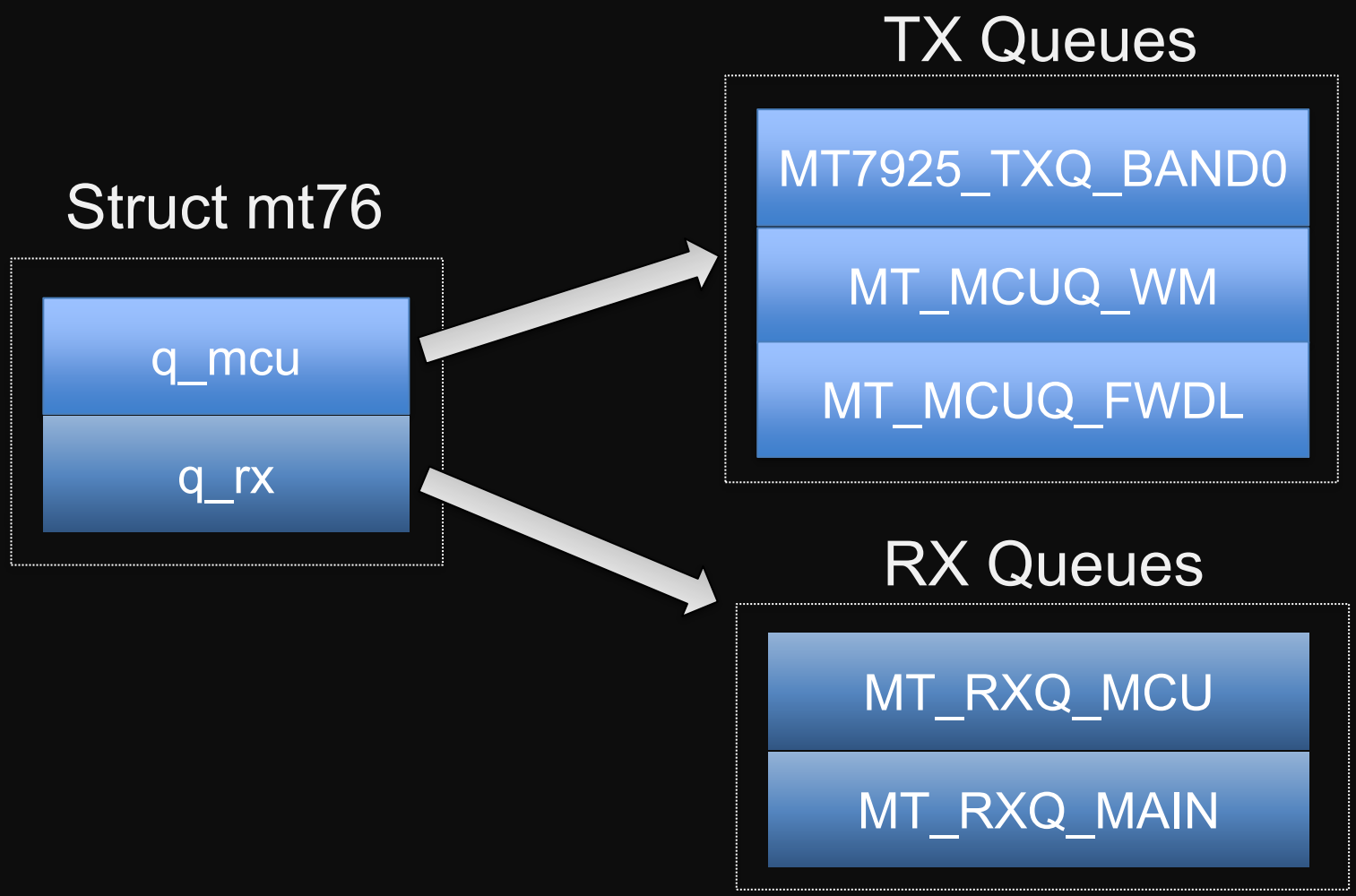
PCIe config

```
!devext 0xffffb981bb35c5c0
PDO Extension, Bus 0x2, Device 0, Function 0.
  DevObj 0xffffb981bb35c470  Parent FDO DevExt 0xffffb981bba56250
  Device State = PciStarted
  Vendor ID 14c3 (MEDIATEK CORP.)  Device ID 7925
  Subsystem Vendor ID 14c3 (MEDIATEK CORP.)  Subsystem ID 7925
  Header Type 0, Class Base/Sub 02/80  (Network Controller/'Other')
  Programming Interface: 00, Revision: 00, IntPin: 01, RawLine 00
  Possible Decodes ((cmd & 7) = 7): BMI
  Capabilities: Ptr=80, power msi express
  Express capabilities: (OS controlled) ltr aer
  Logical Device Power State: D0
  Device Wake Level:          D3
  WaitWakeIrp:                <none>
  Requirements:     Alignment Length    Minimum          Maximum
    BAR0    Mem:    00200000  00200000  0000000000000000 ffffffffffffffff
    BAR2    Mem:    00008000  00008000  0000000000000000 ffffffffffffffff
  Resources:        Start             Length
    BAR0    Mem:    00000000fd000000 00200000
    BAR2    Mem:    00000000fd2f8000 00008000
  Interrupt Requirement:
    Line Based - Min Vector = 0x0, Max Vector = 0xffffffff
    Message Based: Type - Msi, 0x20 messages requested
  Interrupt Resource:    Type - MSI, 0x1 Messages Granted
```

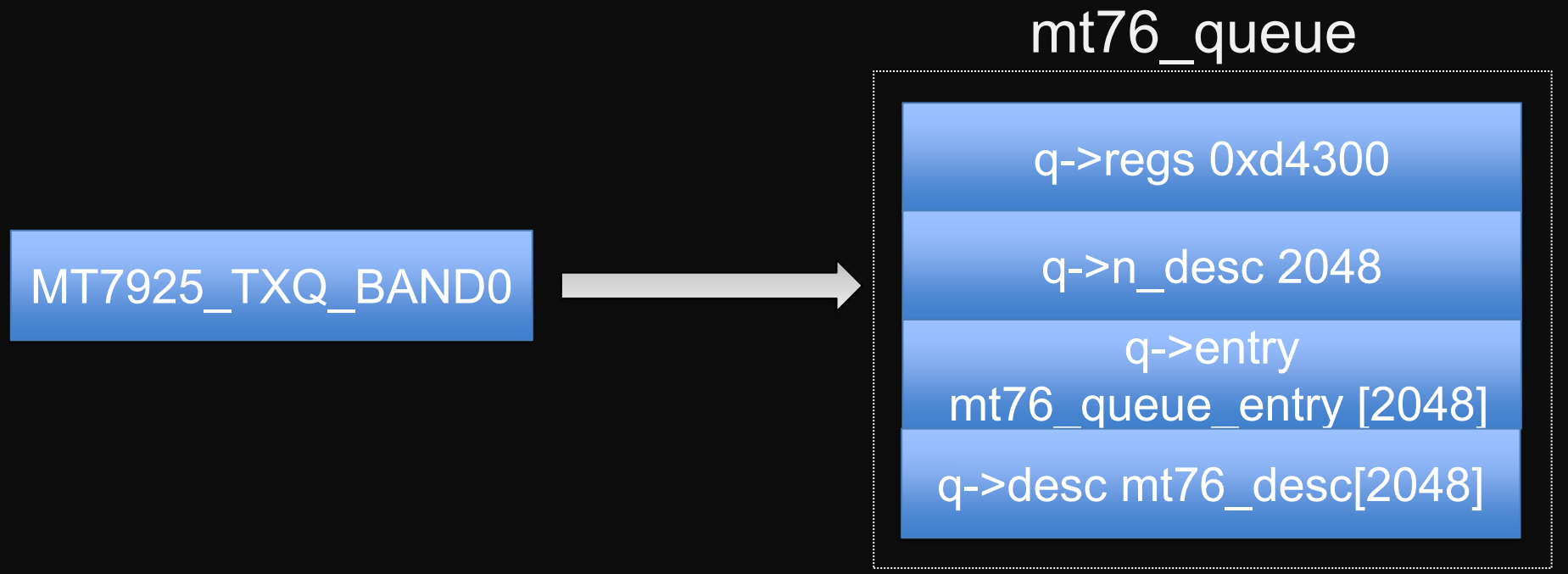
Kernel -> MMIO

- Bus Mastering DMA
- Ring Buffers (circular queues)
  - TX Rings (Transmit)
  - RX Rings (Receive)
  - Control/Cmd Ring
  - Event Ring

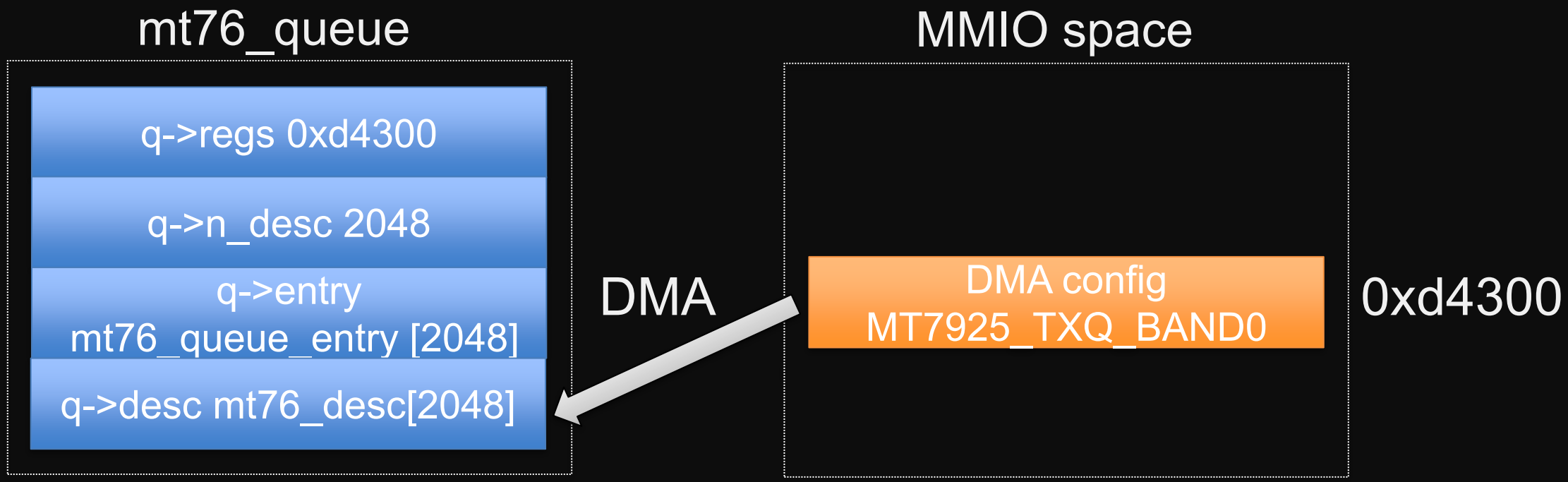
Inside the Kernel



A TX Queue Example



Inside the MMIO





Kernel -> MMIO

```
int64_t AsicConnac3xErrorRecoveryBtL0P4(void* arg1)

    HANDLE arg_18 = nullptr
    int32_t arg_8 = 0
    int32_t Data = 0
    void* trace_enabled_1 = trace_enabled

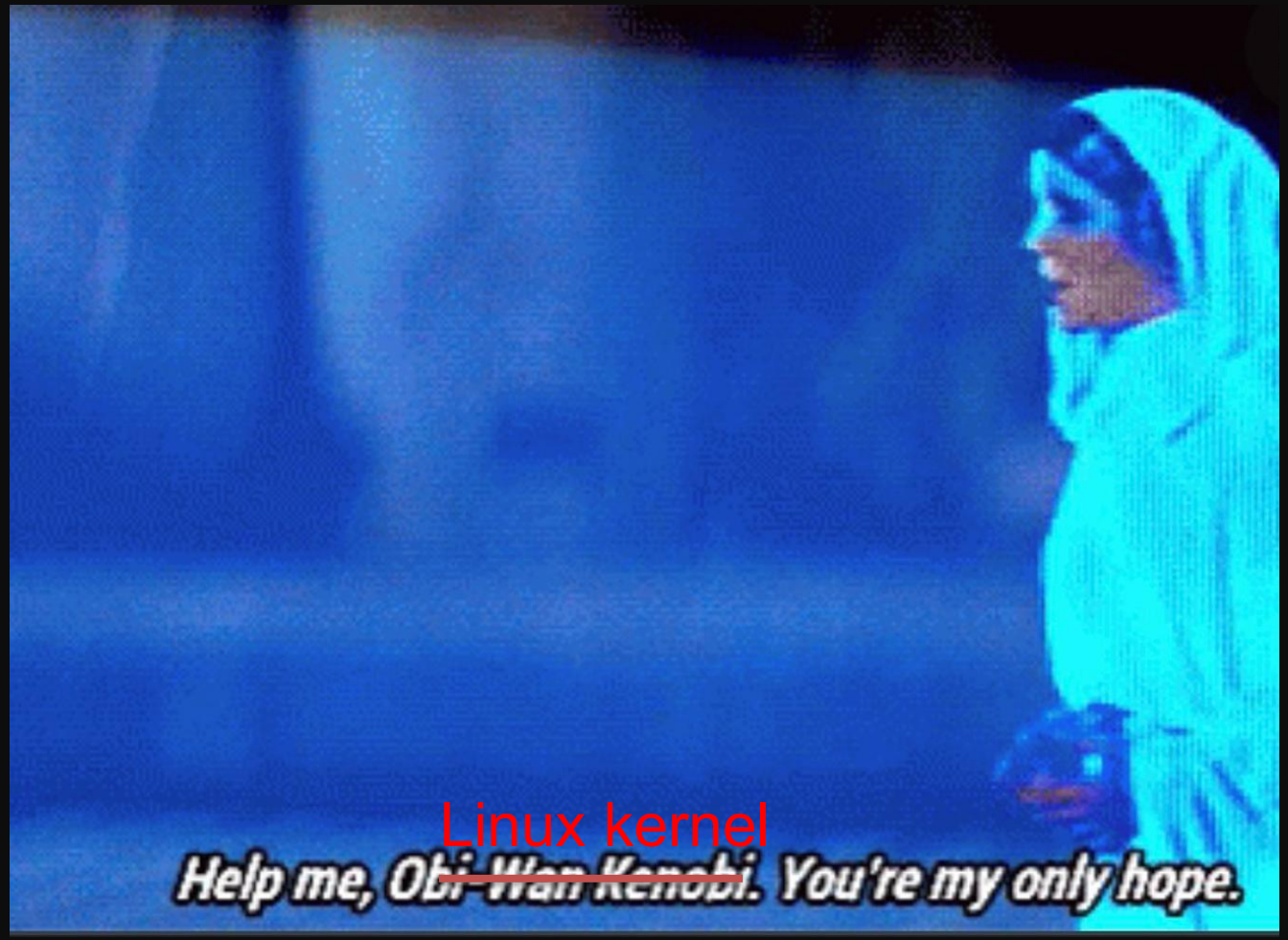
    if (trace_enabled_1 != &trace_enabled && (*(trace_enabled_1 + 0xa4) & 1) != 0
            && *(trace_enabled_1 + 0xa1) u>= 5)
        wmi_trace_wrapper(*(trace_enabled_1 + 0x90), 0x1f4, &data_14021fee0)

    HW_IO_READ32(arg1, 0x70025050, &arg_8)
    int32_t r8_3 = (arg_8 & 0xffffffef) | 8
    arg_8 = r8_3
    HW_IO_WRITE32(arg1, 0x70025050, r8_3)
    HW_IO_READ32(arg1, 0x70028610, &arg_8)
    int32_t r8_7 = (arg_8 & 0xffffffdf) | 1
    arg_8 = r8_7
    HW_IO_WRITE32(arg1, 0x70028610, r8_7)
```

Reads from MMIO

Writes to MMIO

Wtf is 0x70025050?



Linux kernel source

```
static u32 __mt7925_reg_addr(struct mt792x_dev *dev, u32 addr)
{
        static const struct mt76_connac_reg_map fixed_map[] = {
                { 0x830c0000, 0x000000, 0x0001000 }, /* WF_MCU_BUS_CR_REMAP */
                { 0x54000000, 0x002000, 0x0001000 }, /* WFDMA PCIE0 MCU DMA0 */
                { 0x55000000, 0x003000, 0x0001000 }, /* WFDMA PCIE0 MCU DMA1 */
                { 0x56000000, 0x004000, 0x0001000 }, /* WFDMA reserved */
                { 0x57000000, 0x005000, 0x0001000 }, /* WFDMA MCU wrap CR */
                { 0x58000000, 0x006000, 0x0001000 }, /* WFDMA PCIE1 MCU DMA0 (MEM_DMA) */
                { 0x59000000, 0x007000, 0x0001000 }, /* WFDMA PCIE1 MCU DMA1 */
                { 0x820c0000, 0x008000, 0x0004000 }, /* WF_UMAC_TOP (PLE) */
                { 0x820c8000, 0x00c000, 0x0002000 }, /* WF_UMAC_TOP (PSE) */
                { 0x820cc000, 0x00e000, 0x0002000 }, /* WF_UMAC_TOP (PP) */
                { 0x74030000, 0x010000, 0x0001000 }, /* PCIe MAC */
                { 0x820e0000, 0x020000, 0x0000400 }, /* WF_LMAC_TOP BN0 (WF_CFG) */
                { 0x820e1000, 0x020400, 0x0000200 }, /* WF_LMAC_TOP BN0 (WF_TRB) */
                { 0x820e2000, 0x020800, 0x0000400 }, /* WF_LMAC_TOP BN0 (WF_AGG) */
                { 0x820e3000, 0x020c00, 0x0000400 }, /* WF_LMAC_TOP BN0 (WF_ARB) */
                { 0x820e4000, 0x021000, 0x0000400 }, /* WF_LMAC_TOP BN0 (WF_TMAC) */
                { 0x820e5000, 0x021400, 0x0000800 }, /* WF_LMAC_TOP BN0 (WF_RMAC) */
                { 0x820ce000, 0x021c00, 0x0000200 }, /* WF_LMAC_TOP (WF_SEC) */
                { 0x820e7000, 0x021e00, 0x0000200 }, /* WF_LMAC_TOP BN0 (WF_DMA) */
                { 0x820cf000, 0x022000, 0x0001000 }, /* WF_LMAC_TOP (WF_PF) */
                { 0x820e9000, 0x023400, 0x0000200 }, /* WF_LMAC_TOP BN0 (WF_WTBLOFF) */
                { 0x820ea000, 0x024000, 0x0000200 }, /* WF_LMAC_TOP BN0 (WF_ETBF) */
```

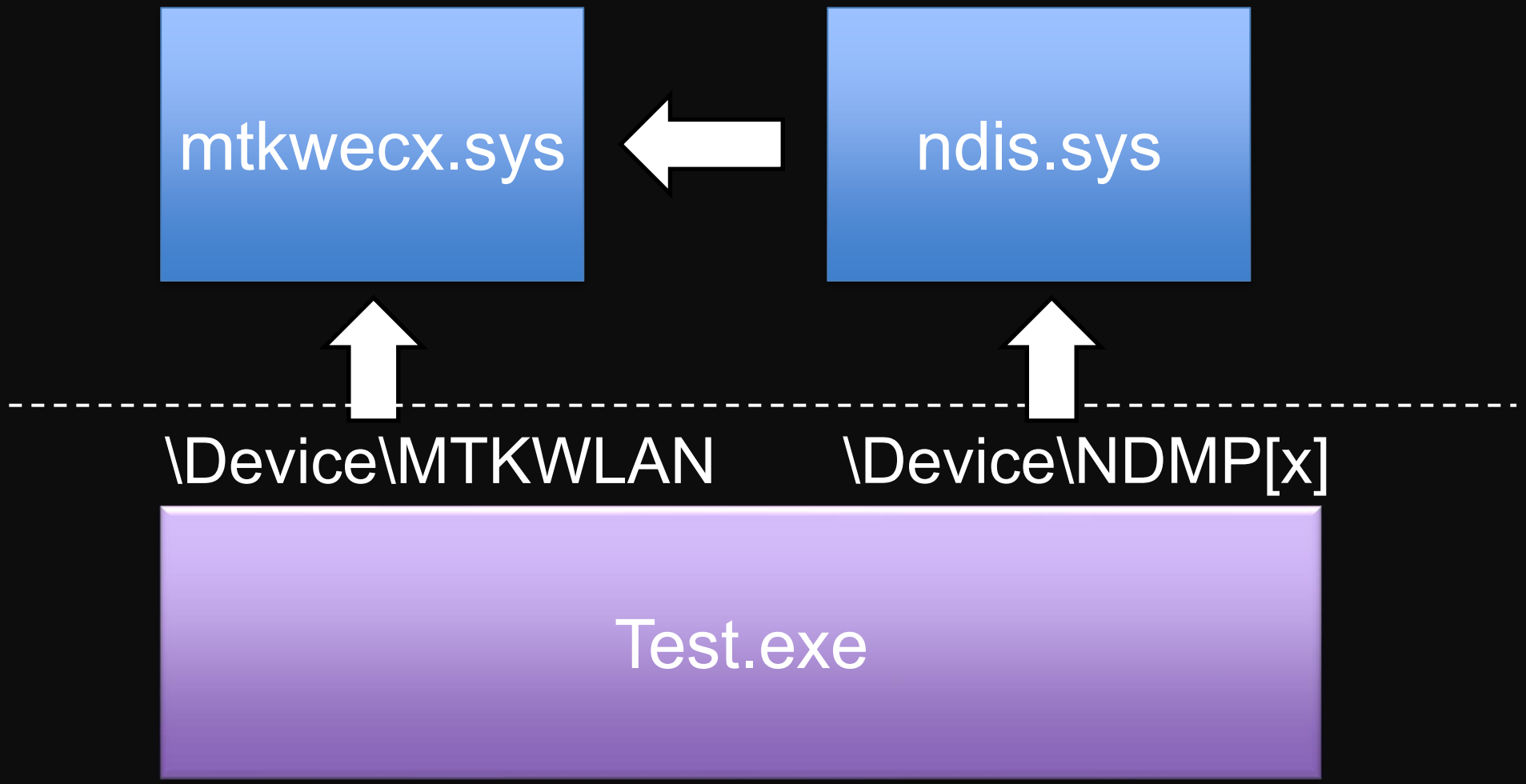
https://github.com/torvalds/linux/blob/master/drivers/net/wireless/mediatek/mt76/mt7925/pci.c

Cracking the firmware

User mode Kernel mode Firmware



Usermode -> Kernel



https://github.com/IOActive/FuzzNDIS/blob/master/Attacking_Windows_NDIS_Drivers.pdf

Setting io_mode

```
int64_t FuncSetIoMode(struct mt_big_struct* arg1, void* arg2, void* mappedVA, int32_t size,

    int32_t* rax = out_size.q

    if (size u< 4)  // input size has to be bigger than 4
        *rax = 0
        *out_size_ptr = 4
        return 0x80000005

    *rax = 4
    *out_size_ptr = 4
    void* rcx = data_14022b148

    // Checks if logging is enabled and logs the function call
    if (rcx != &data_14022b148 && (*(rcx + 0x4a0) & 1) != 0 && *(rcx + 0x49d) u>= 3)
        int32_t var_18_1 = *mappedVA
        logger(*(rcx + 0x48c), 0x19f, &data_140215900, "FuncSetIoMode")

    // here the io_mode gets set to a userdefined value
    arg1->io_mode = *mappedVA
    return 0
```

Sets io_mode

MMIO access?

```
int64_t FuncSetCr(struct mt_big_struct* arg1, void* arg2, void* mappedVA, int32_t size,

    int32_t* rax = out_size.q
    void* mappedVA_1 = mappedVA

    if (size u< 8)
        *rax = 0
        *out_size_ptr = 8
        return 0x80000005

    *rax = 8
    *out_size_ptr = 8
    void* rcx = data_14022b148

    if (rcx != &data_14022b148 && (*(rcx + 0x4a0) & 1) != 0 && *(rcx + 0x49d) u>= 3)
        char var_40
        var_40.d = zx.d(KeGetCurrentIrql())
        char var_48
        var_48.d = *mappedVA_1
        logger2(*(data_14022b148 + 0x48c), 0x72, &data_140215900, "FuncSetCr")

    uint64_t address = zx.q(*mappedVA_1)

    // if ioMode is == 1 we access the MMIO directly
    if (arg1->io_mode == 1 || (address.d & 0xfff00000) == 0)
        HW_IO_WRITE32(arg1, address.d & 0xfffffffc, *(mappedVA_1 + 4))
```

From usermode

CVE-2026-20423

https://euvd.enisa.europa.eu/vulnerability/CVE-2026-20423

Triggering a coredump

```
uint64_t PlatformOpenCoreDumpFile(struct mt_big_struct* arg1, char arg2)

    NTSTATUS rbx_1

    if (arg2 == 0)
        rbx_1 = ZwClose(Handle: arg1->filehandle_dump)
    else
        wchar16* SourceString

        if (arg1->field_188941d == 0)
            SourceString = u"\SystemRoot\System32\drivers\HostPcieCoreDump"

            if (arg1->field_188941c == 0)
                SourceString = u"\SystemRoot\System32\drivers\CoreDump"
        else
            SourceString = u"\SystemRoot\System32\drivers\CoreDumpBT"

        UNICODE_STRING DestinationString
        RtlInitUnicodeString(&DestinationString, SourceString)
        OBJECT_ATTRIBUTES ObjectAttributes
        ObjectAttributes.RootDirectory = 0
        int64_t var_98_1 = 0
        ObjectAttributes.Length = 0x30
        ObjectAttributes.Attributes = 0x240
        ObjectAttributes.ObjectName = &DestinationString
        ObjectAttributes.SecurityDescriptor = 0
        ObjectAttributes.SecurityQualityOfService = 0
        IO_STATUS_BLOCK IoStatusBlock
        NTSTATUS rax = ZwCreateFile(FileHandle: &arg1->filehandle_dump,
            DesiredAccess: 0xc0000000, &ObjectAttributes, &IoStatusBlock,
            AllocationSize: nullptr, FileAttributes: 0x80, ShareAccess: 1,
            CreateDisposition: 3, CreateOptions: 0x860, EaBuffer: nullptr, EaLength: 0)
        rbx_1 = rax
```

Overwrite MMIO
until file is created

Inside the coredump

RiscV registers

```
; registers
R.S PC 0x4
R.S T0 0x2221000
R.S T1 0x41414141
R.S T2 0x0
R.S S0 0x64
R.S S1 0x0
R.S A0 0xE04
R.S A1 0xCF759
R.S A2 0x0
R.S A3 0x64
R.S A4 0x22142E8
R.S A5 0xE0006000
R.S A6 0x221C0C8
R.S A7 0x15
R.S S2 0x828000
R.S S3 0x2235000
R.S S4 0x828000
R.S S5 0x2233000
R.S S6 0x828000
R.S S7 0x828000
R.S S8 0x828000
R.S S9 0xE02
R.S S10 0xA5A5A5A5
R.S S11 0xA5A5A5A5
R.S T3 0x0
R.S T4 0x22363B8
```

Stacktrace

```
PC log contorl=16FFFC(T=0xCFA6C)
PC log(0)=0xE00355A6
PC log(1)=0x90DE2A
PC log(2)=0x90DE2A
PC log(3)=0x90D686
PC log(4)=0x90D692
PC log(5)=0x84A36C
PC log(6)=0x84A376
PC log(7)=0x80E3F2
PC log(8)=0x80E3F8
PC log(9)=0x84A37C
PC log(10)=0x84A388
PC log(11)=0x90D694
PC log(12)=0x90D6D4
PC log(13)=0xE0006004
PC log(14)=0xE000601E
PC log(15)=0x80C900
PC log(16)=0x80C950
```


But Wait...
There's
MORE!

The coredump function

```
void* const* mt7925_host_coredump(void* arg1)

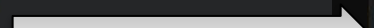
1401d99bd    if (trace_enabled_3 != &trace_enabled) ...
1401d99bd
1401d9a6f    int32_t rax = r14_1[-1]
1401d9a78    int32_t rax_1
1401d9a78    char arg_8
1401d9a78
1401d9a78    // Dumps three undocumented memory regions 0x2200000,
1401d9a78    // 0x400000, 0xe0000000
1401d9a78    if (rax != 0x2200000)
1401d9af0        if (rax != 0x400000)
1401d9b5b            HW_IO_WRITE32(arg1, 0x7c400120, 0xe0000000)
1401d9b6b            HW_IO_WRITE32(arg1, 0x830c0120, 0xe0000000)
1401d9b7b            HW_IO_READ32(arg1, 0xe0000000, &arg_10)
1401d9ba4            dump_region(arg1, 0xe0000000,
1401d9ba4                memory_regions[rbp].size - 0x20000000)
1401d9bae            arg_8 = 0
1401d9bc7            result = MTEnqueueInternalCmd(arg1, r15, 0x10000041, &arg_8, 1)
1401d9bc7            break
1401d9bc7
1401d9af8        HW_IO_WRITE32(arg1, 0x7c400120, 0x400000)
1401d9b0b        HW_IO_WRITE32(arg1, 0x830c0120, 0x400000)
1401d9b1d        HW_IO_READ32(arg1, 0x400000, &arg_10)
1401d9b22        rax_1 = 0x400000
```

Not in the linux source

It’s a code region

Coredump

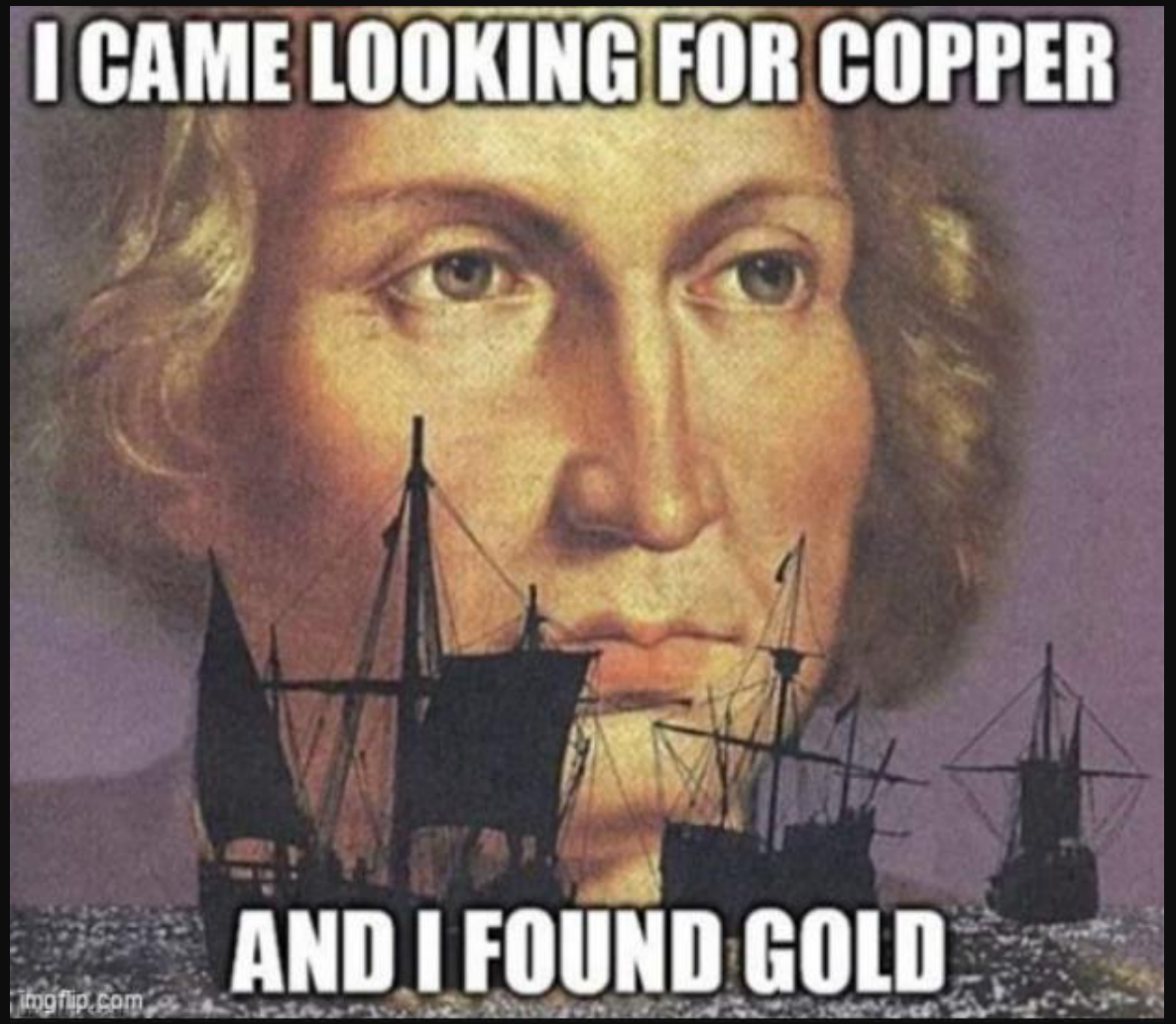
```
HW_IO_WRITE32(arg1, 0x7c400120, 0xe0000000)
HW_IO_WRITE32(arg1, 0x830c0120, 0xe0000000)
HW_IO_READ32(arg1, 0xe0000000, &arg_10)
dump_region(arg1, 0xe0000000,
    memory_regions[rbp].size - 0x20000000)
arg_8 = 0
result = MTEnqueueInternalCmd(arg1, r15, 0x10000041, &arg_8, 1)
break
```



Stacktrace

```
PC log contorl=16FFFC(T=0xCFA6C)(latest PC index = 0x16)
PC log(0)=0xE00355A6
PC log(1)=0x90DE2A
PC log(2)=0x90DE2A
PC log(3)=0x90D686
```




I CAME LOOKING FOR COPPER
AND I FOUND GOLD
imgflip.com

But…

Writing into the code region triggers an assert

Code execution reliability: 20%

Only small code snippets possible

Exploring the Firmware

Control and Status Register (CSR) are registers used for reading status and changing configuration

https://riscv.github.io/riscv-isa-manual/snapshot/privileged/#_currently_allocated_risc_v_machine_level_csr_addresses

Exploring the Firmware

mstatus CSR register

```
PER.S CSR:0x300 0x1880   ; mstatus
```

Machine mode

Shellcode

```
csrr t0, mvendorid
csrr t1, misa
li  s6, 0x0
jr  s6          # Crashes so Coredump triggers
```

Gets JEDEC ID of the vendor
Gets the supported ISA

https://riscv.github.io/riscv-isa-manual/snapshot/privileged/#_currently_allocated_risc_v_machine_level_csr_addresses

Exploring the Firmware

mvendorid

Resolved to AndesTech

misa

```
misa = 0x40941105
MXL (ISA width): 1 → RV32

Implemented extensions:
  A : Atomic instructions
  C : Compressed instructions
  I : Base integer ISA
  M : Integer multiply/divide
  S : Supervisor mode implemented
  U : User mode implemented
  X : Non-standard extensions
```



???

https://riscv.github.io/riscv-isa-manual/snapshot/privileged/#_machine_vendor_id_mvendorid_register

https://riscv.github.io/riscv-isa-manual/snapshot/privileged/#misa

Exploring the Firmware

AndeStar™ V5

- Custom RiscV extension
- Includes CoDense™ instructions
- GP relative loading
- Custom jumps

https://www.andestech.com/en/products-solutions/product-documentation/

Exploring the Firmware

AndeStar™ V5

```
e00b0bbc  4091       nexec.it 0x9104d4
e00b0bbe  2a84       mv       s0, a0
e00b0bc0  ae84       mv       s1, a1
e00b0bc2  81e5       bnez     a1, data_e00b0bca

e00b0bc4  8d47       li       a5, 0x3
e00b0bc6  63f9a700   bgeu     a5, a0, data_e00b0bd8

e00b0bca  aaaf       fsd      ft11, 0x88(sp) {arg_88}
e00b0bcc  0146       li       a2, 0x0
e00b0bce  93850584   addi     a1, a1, -0x7c0
e00b0bd2  46a0       fsd      ft0, 0x50(sp) {arg_50}
e00b0bd4  ef7079d2   jal      ra, sub_e00488fa

e00b0bd8  11cc       beqz     s0, data_e00b0bf4

e00b0bda  5b511402   beqc     s0, 0x1, 0xe00b0bfc
```

https://www.andestech.com/en/products-solutions/product-documentation/
https://2025.hexacon.fr/slides/Arise_from_the_Wireless-Breaking_the_Security_Barrier_in_Wi-Fi.pdf

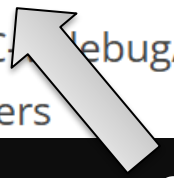
Exploring the Firmware

AndeStar™ V5

The Key Features of V5 Architecture

RISC-V Compliance such as:

- RV32/64IMAC[FDPBKZc] ([]: depends on CPU cores and configurable options)
- Vector Extension (RVV)
- RVA profile oriented for AP
- PLIC and CLIC
- MMU for Linux applications
- PMP for TEE isolation
- RISC- ebug/trace standard
- Others



???

https://www.andestech.com/en/products-solutions/andestar-architecture/

Exploring the Firmware

Physical Memory Protection (PMP)

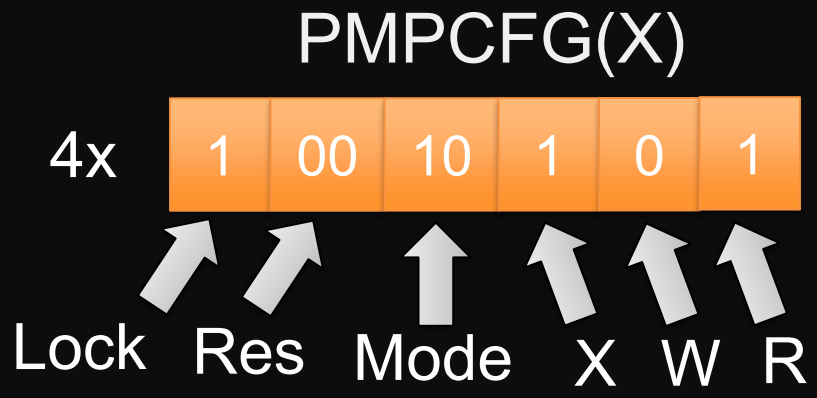
Sets R/W/X permissions per memory regions

No MMU needed

https://www.andestech.com/en/products-solutions/andestar-architecture/

Exploring the Firmware

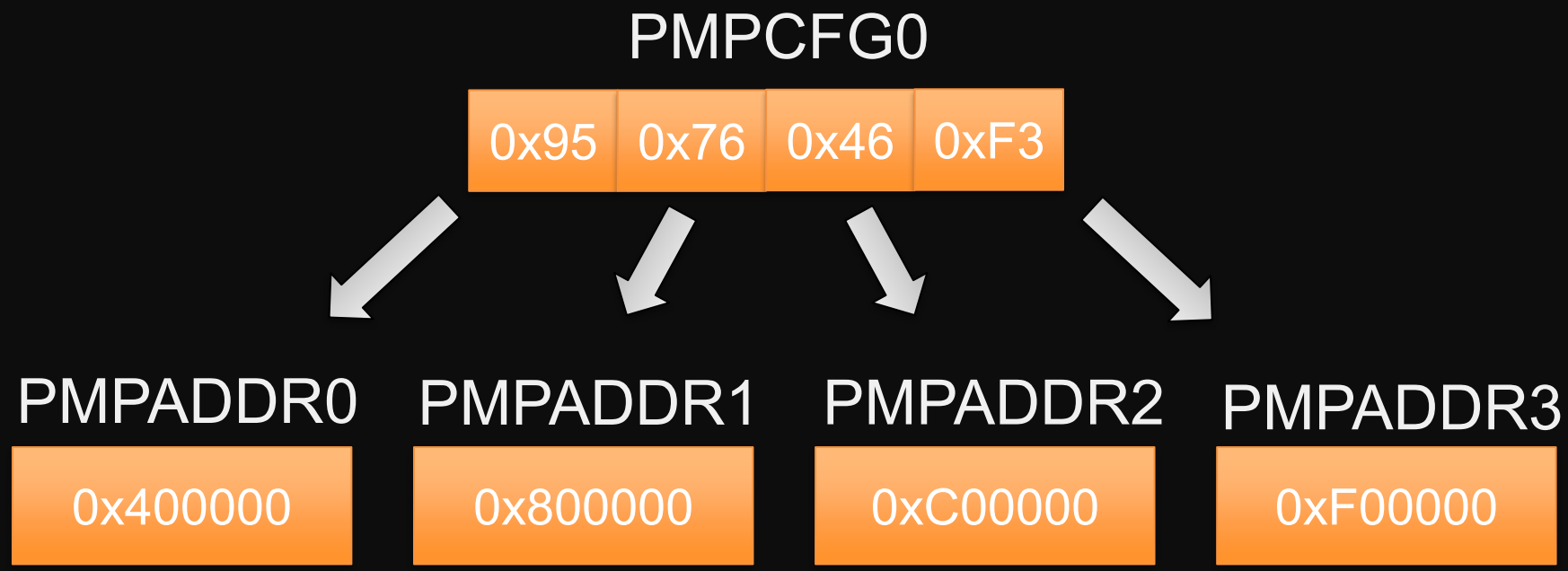
Physical Memory Protection (PMP)



https://riscv.github.io/riscv-isa-manual/snapshot/privileged/#_physical_memory_protection_csrs

Exploring the Firmware

Physical Memory Protection (PMP)



https://riscv.github.io/riscv-isa-manual/snapshot/privileged/#_physical_memory_protection_csrs

Memory regions

Mode	Region	Permission	Used for	Locked
TOR	0x400000 - 0x41C000	RW	SRAM?	Y
TOR	0x800000 - 0x920000	RX	ROM + ROM Patch	Y
TOR	0x2200000 - 0x2270000	RW	Stack + Global	Y
TOR	0x50000000 - 0xC0000000	RW	???	Y
TOR	0xE0000000 - 0xE0006800	RWX	Code(RAM)	Y
TOR	0xE0006800 - 0xE000A800	RW	Data(RAM)	Y
TOR	0xE000A800 - 0xE0020C00	RX	Code(RAM)	Y
TOR	0xE0020C00 - 0xE002D000	RW	Data(RAM)	Y
TOR	0xE002D000 - 0xE04A0000	RX	Code(RAM)	Y
TOR	0xE04A0000 - 0xFFFFFFF8	RWX	Undefined	Y

Upgrading the code injection

1. Write shellcode into 0xE0006000
2. Overwrite function pointer at 0x22142e8
3. ???
4. Profit (100% reliable)

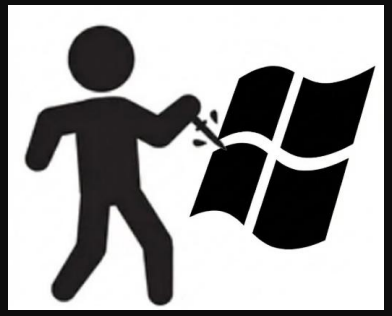
Firmware code exec. For what?



Tracking



Evasion



Backstab the kernel

https://berlin2025.nullcon.net/berlin-2025/speaker-speaker-unlock-hidden-superpowers

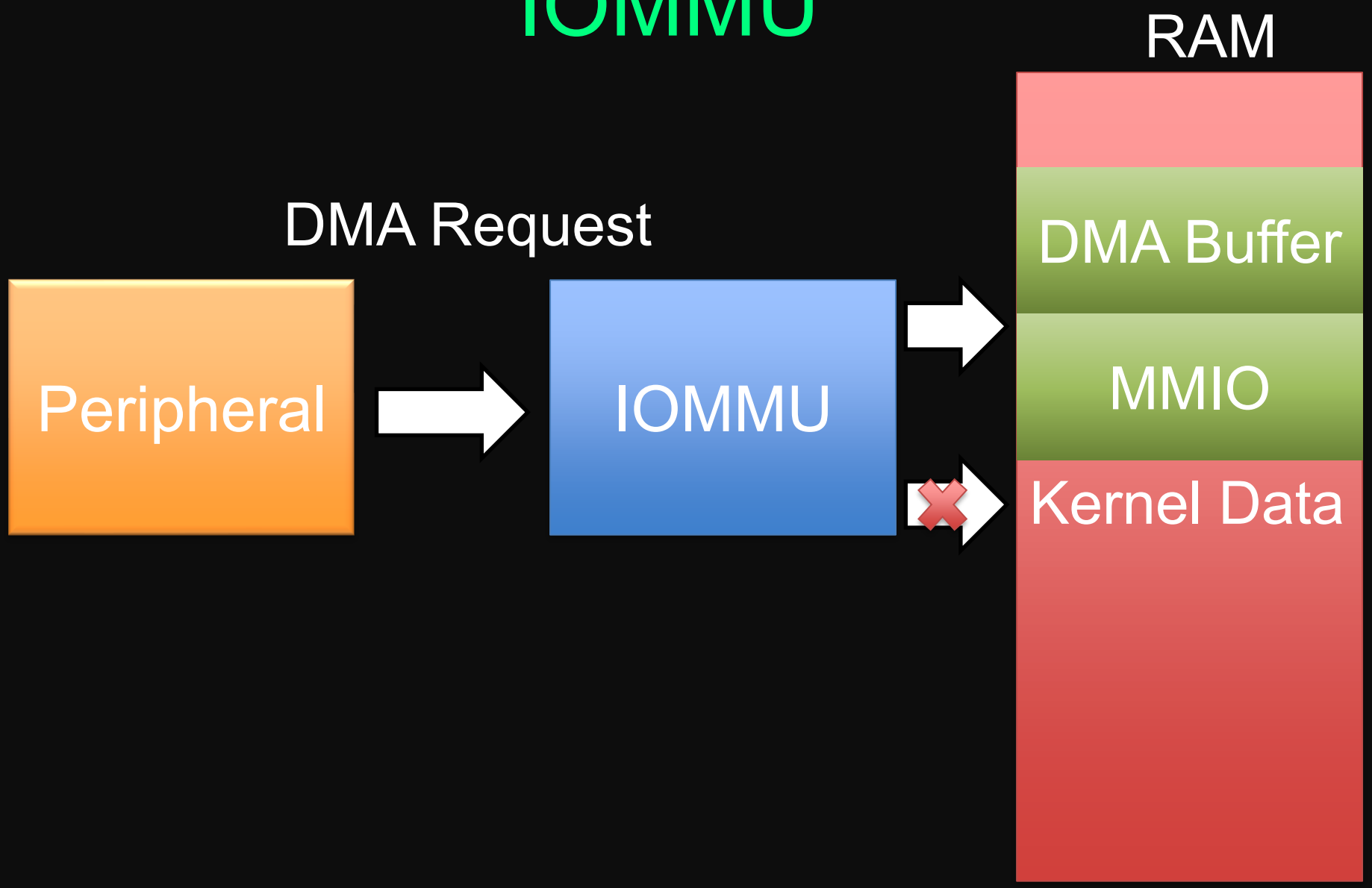
Returning to the kernel

User mode Kernel mode Firmware



Mitigations?

IOMMU


RAM
DMA Request
Peripheral
IOMMU
DMA Buffer
MMIO
Kernel Data


IOMMU IS A GOOD ENOUGH PROTECTION
SURE GRANDMA LET'S GET YOU TO BED

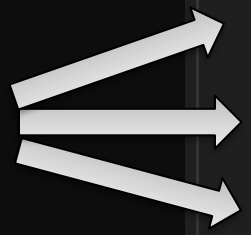
IOMMU just filters address access

Data from the firmware still needs to be parsed

But where is the data parsed?

receive queues

Parsers



```
switch (type) {
case PKT_TYPE_TXRX_NOTIFY:
    /* PKT_TYPE_TXRX_NOTIFY can be received only by mmio devices */
    mt7925_mac_tx_free(dev, skb->data, skb->len);
    napi_consume_skb(skb, 1);
    break;
case PKT_TYPE_RX_EVENT:
    mt7925_mcu_rx_event(dev, skb);
    break;
case PKT_TYPE_TXS:
    for (rxd += 2; rxd + 8 <= end; rxd += 8)
        mt7925_mac_add_txs(dev, rxd);
    dev_kfree_skb(skb);
    break;
case PKT_TYPE_NORMAL_MCU:
case PKT_TYPE_NORMAL:
    if (!mt7925_mac_fill_rx(dev, skb)) {
        mt76_rx(&dev->mt76, q, skb);
        return;
    }
    fallthrough;
```

https://github.com/torvalds/linux/blob/v6.17-rc4/drivers/net/wireless/mediatek/mt76/mt7925/mac.c#L1203



RX_EVENT

```c
void mt7925_mcu_rx_event(struct mt792x_dev *dev, struct sk_buff *skb)
{
    struct mt7925_mcu_rxd *rxd = (struct mt7925_mcu_rxd *)skb->data;

    if (skb_linearize(skb))
        return;

    if (rxd->option & MCU_UNI_CMD_UNSOLICITED_EVENT) {
        mt7925_mcu_uni_rx_unsolicited_event(dev, skb);
        return;
```

👀

https://github.com/torvalds/linux/blob/v6.17-rc4/drivers/net/wireless/mediatek/mt76/mt7925/mcu.c#L559

UNSOLICITED EVENTS



```c
static void
mt7925_mcu_uni_rx_unsolicited_event(struct mt792x_dev *dev,
                                    struct sk_buff *skb)
{
    struct mt7925_mcu_rxd *rxd;

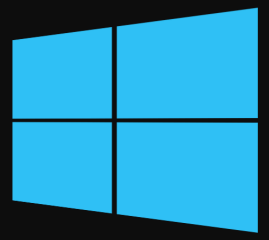
    rxd = (struct mt7925_mcu_rxd *)skb->data;

    switch (rxd->eid) {
    case MCU_UNI_EVENT_HIF_CTRL:
        mt7925_mcu_uni_hif_ctrl_event(dev, skb);
        break;
    case MCU_UNI_EVENT_FW_LOG_2_HOST:
        mt7925_mcu_uni_debug_msg_event(dev, skb);
        break;
    case MCU_UNI_EVENT_ROC:
        mt7925_mcu_uni_roc_event(dev, skb);
        break;
    case MCU_UNI_EVENT_SCAN_DONE:
        mt7925_mcu_scan_event(dev, skb);
        return;
    case MCU_UNI_EVENT_TX_DONE:
        mt7925_mcu_tx_done_event(dev, skb);
        break;
    case MCU_UNI_EVENT_BSS_BEACON_LOSS:
```

8 Events

https://github.com/torvalds/linux/blob/v6.17-rc4/drivers/net/wireless/mediatek/mt76/mt7925/mcu.c#L523

UNSOLICITED EVENTS



```
char RamModeUniEventHandler(void* arg1, uint16_t* __offset(mt7925_mcu_rxd, 0x
140144b9e     int64_t rbx
140144b9e
140144b9e     while (true)
140144b9e         if (rcx_2->nr == eid.w)
140144c5e             uint16_t* __offset(mt7925_mcu_rxd, 0x10)* __offset(m
140144c5e                 event = arg2
140144c5e
140144c61             if (eid.b != 0x7c)
140144c73                 rbx.b = 1
140144c7d                 g_unieventhandlers[rax_2].func(arg1, event)
140144c61             else
140144c6b                 rbx.b = nicUniEventFactCal(arg1.b, event)
140144c6b
```

```
struct uni_event_handler g_unieventhandlers[0x21] =
{
    [0x00] =
    {
        uint16_t eid = 0x1
        void (* func)() = nicUniCmdEventResultToHif
    }
    [0x01] =
    {
        uint16_t eid = 0x43
        void (* func)() = nicUniCmdEventQueryNicCapabilityV2
    }
    [0x02] =
    {
        uint16_t eid = 0x4
        void (* func)() = nicUniEventFwLog2Host
    }
    [0x03] =
```

34 Events

nicUniEventFactCal

```
140147cc4    bool nicUniEventFactCal(char arg1, struct mt7925_mcu_rxd_cal_event* event)

140147cd1        arg1 = 1
140147cd1
140147cd9        if (event->field_10 == 0)
140147cdf            bool set_iterator_1 = event->set_iterator
140147ce4            int64_t rdi
140147ce4            rdi.b = set_iterator_1 == 1
140147d04            factCalAddData(event->field_18 != 0, event_0x20: event->must_be_1,
140147d04                event_0x14: event->field_14, src_data: &event->data, event->size,
140147d04                set_iterator: set_iterator_1)
140147d09            void* trace_enabled_1 = trace_enabled
140147d17            arg1 = rdi.b
```

Fully from the FW

factCalAddData

```
int64_t factCalAddData(char arg1, char event_0x20, int32_t event_0x14, int64_t* src_data, int32_t size, char set_iterator)

    uint32_t event_0x20_1 = zx.d(event_0x20)
    uint32_t rbx = zx.d(arg1)
    void* trace_enabled_1 = trace_enabled

    if (trace_enabled_1 != &trace_enabled && (*(trace_enabled_1 + 0x2c) & 1) != 0
        && *(trace_enabled_1 + 0x29) u>= 3) ...

    if (rbx.b != 0)
        if (rbx != 1) ...

        if (event_0x20_1.b u<= 1)
            int64_t current_cal_offset = zx.q(cal_iter) * 0x17e4
            // If we call the function the second time
            // cal_data_size will be overflowed by us
            memcpy(
                zx.q((&cal_data_size)[zx.q(cal_iter) * 0x5f9]) + current_cal_offset +
                    &cal_data,
                src_data, size)
            (&cal_data_size)[zx.q(cal_iter) * 0x5f9] += size

            // with the first call we have to set the iterator
            if (set_iterator != 0)
                (&data_1402448f1)[current_cal_offset] = 1
                int32_t iter_plus_1 = zx.d(cal_iter) + 1
                cal_iter = iter_plus_1.b - (zx.q(iter_plus_1 u/ 0xa) * 5).b * 2
    else if (event_0x20_1.b u<= 1) ...

    return 0
```

Coming from FW

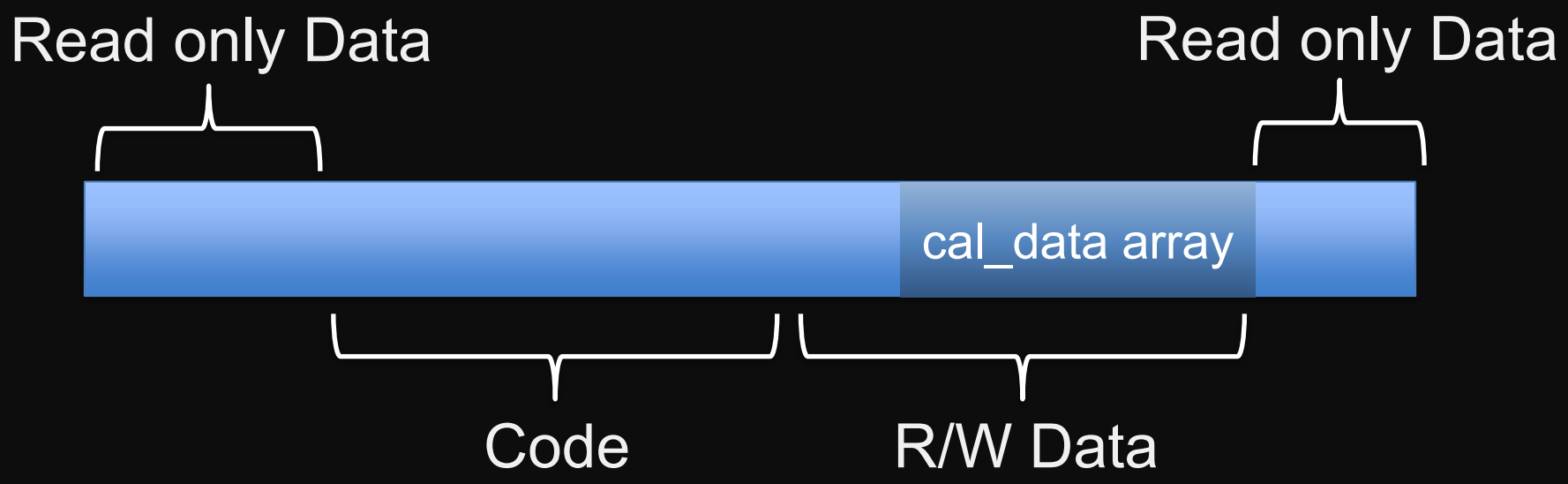
No bounds check

Increment Iterator

CVE-2026-20436

https://euvd.enisa.europa.eu/vulnerability/CVE-2026-20436

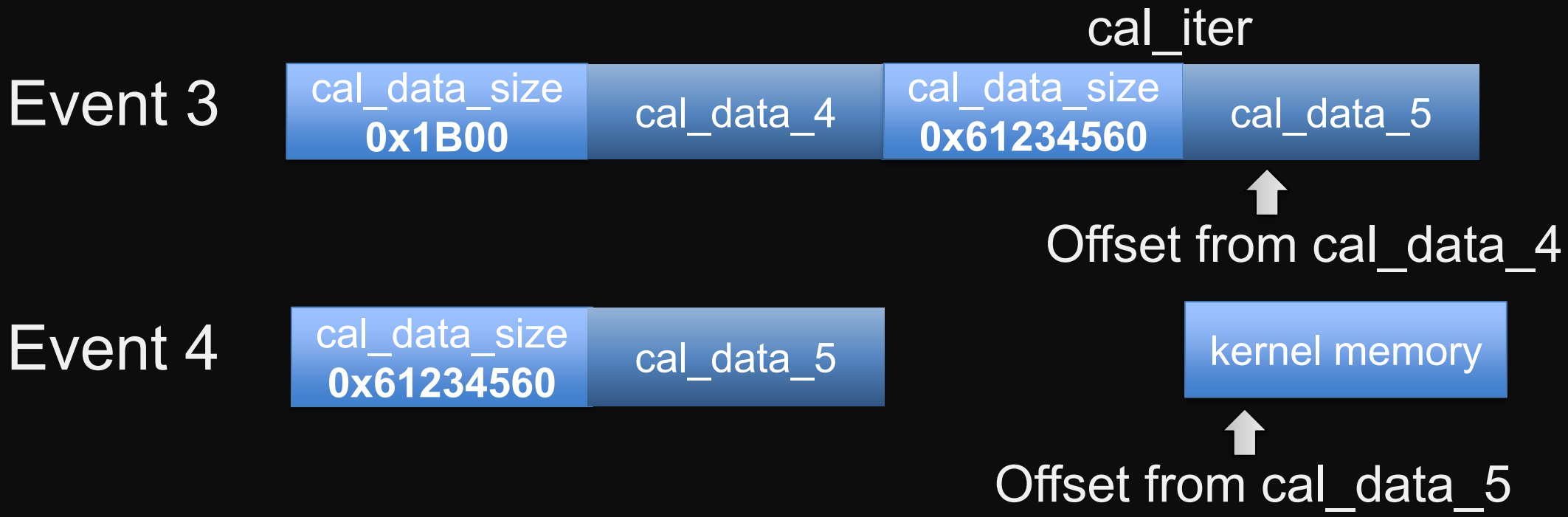
mtkwecx.sys



CalData


cal_iter
Start
cal_data_size
0x0
cal_data_4
cal_data_size
0x0
cal_data_5
Event 1
cal_data_size
0x900
cal_data_4
cal_data_size
0x0
cal_data_5
Offset from
cal_data_4
Event 2
cal_data_size
0x1200
cal_data_4
cal_data_size
0x0
cal_data_5
Offset from
cal_data_4

CalData



https://devco.re/blog/2024/10/05/streaming-vulnerabilities-from-windows-kernel-proxying-to-kernel-part2-en/

Time to leak

User mode Kernel mode Firmware



Obtaining address of mtkwecx.sys

```
uint64_t FuncGetWFRF(struct mt_big_struct* arg1, void* p_userdata)

    int32_t field_14744f8 = arg1->field_14744f8
    int32_t i_1 = 0
    *(p_userdata + 0x20) = field_14744f8
    int32_t i = 0

    if (field_14744f8 != 0) ...

    uint64_t iterator = zx.q(arg1->raceable)
    // writes iterator to userspace
    *(p_userdata + 0x78) = iterator.d

    if (iterator.d != 0)
        do
            uint64_t rcx_1 = zx.q(i_1) * 2
            int64_t rax_6 = (zx.q(i_1) + 0x50) * 2
            i_1 += 1
            *(p_userdata + (rcx_1 << 3) + 0x80) = *(arg1 + (rax_6 << 3) + 0x1474058)
            iterator = *(arg1 + (rcx_1 << 3) + 0x1474560)
            *(p_userdata + (rcx_1 << 3) + 0x88) = iterator
        while (i_1 u< *(p_userdata + 0x78))

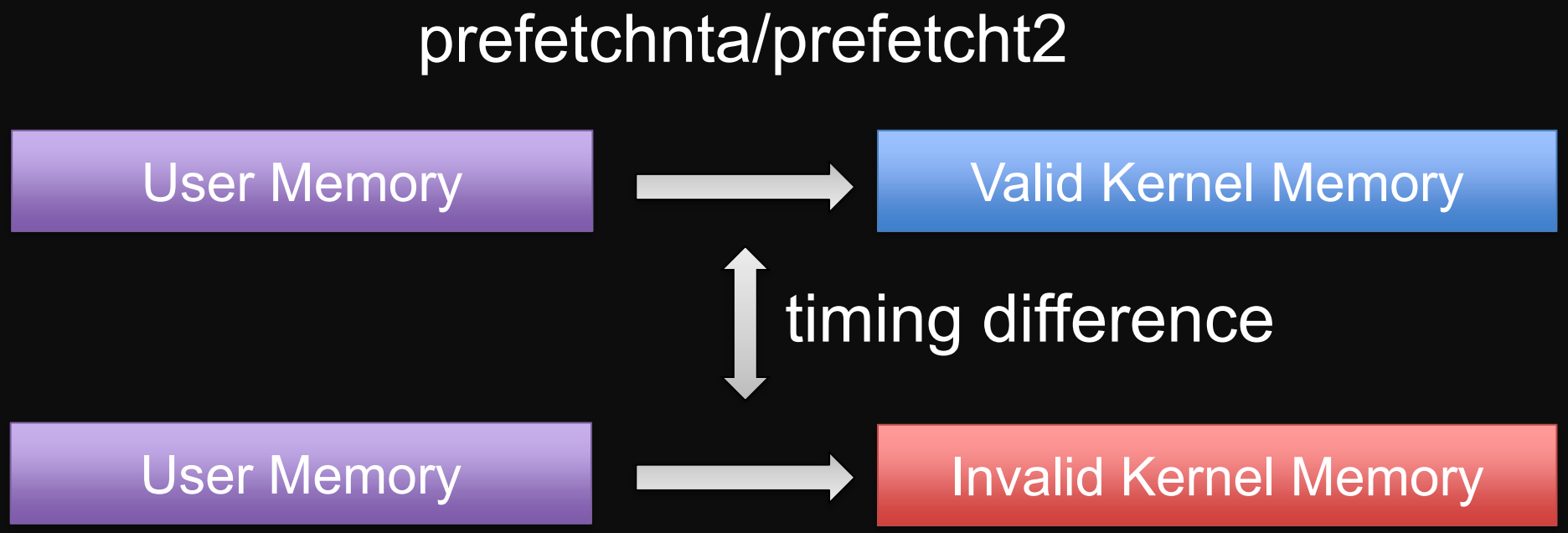
    return iterator
```

Resides in userspace

CVE-2025-20740

https://euvd.enisa.europa.eu/vulnerability/CVE-2025-20740

Prefetch side channel



https://gruss.cc/files/prefetch.pdf

Getting address of nt

Not reliable on AMD CPUs

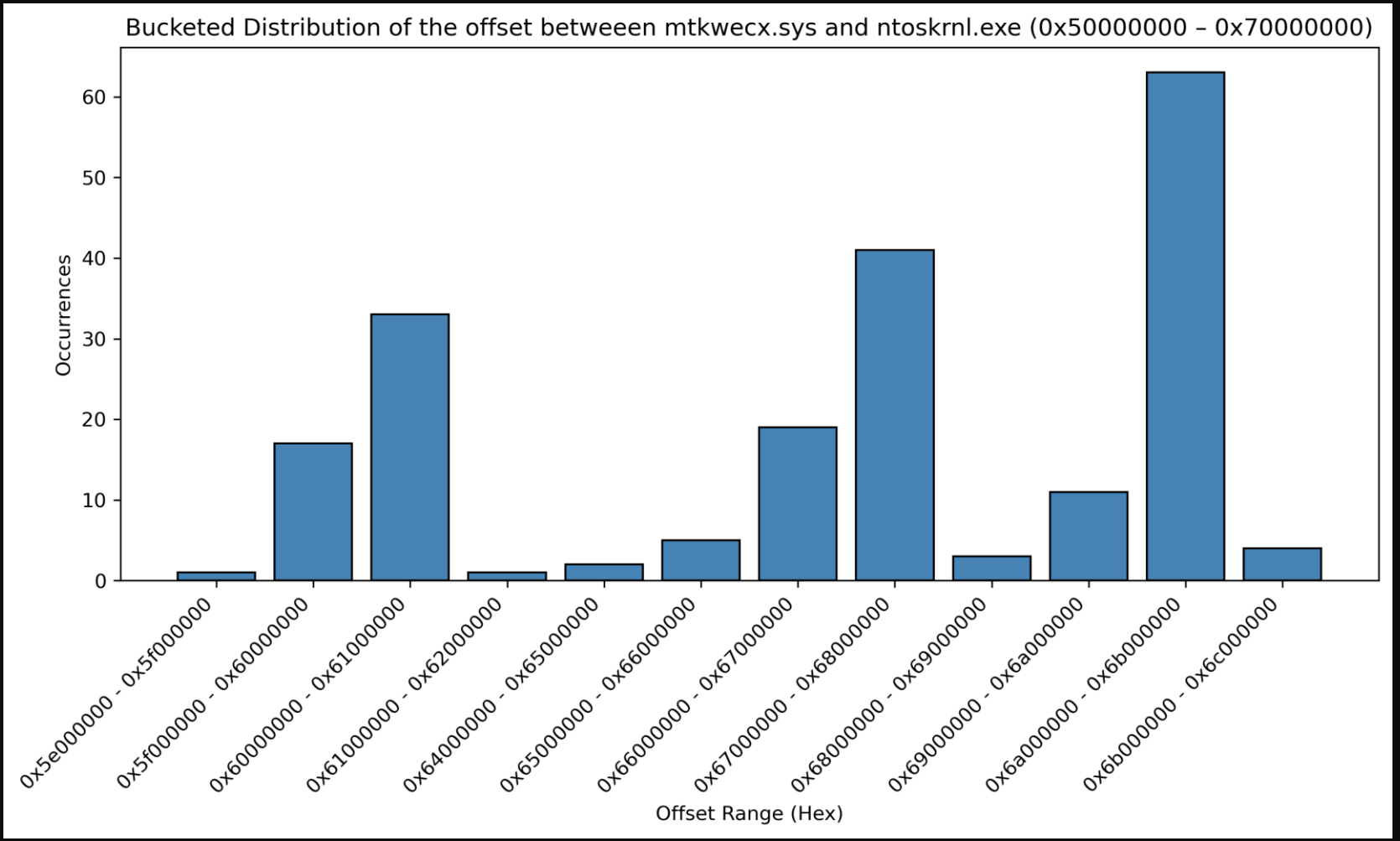


What do we have?



An address inside mtkwecx!

Distance mt -> nt



Measured over 200 reboots

Prefetch search space reduction

Before:

$$\frac{0xfffff80800000000 - 0xfffff80000000000}{0x100000} = 0x8000$$

After:

$$\frac{(mt + 0x70000000) - (mt + 0x50000000)}{0x100000} = 0x200$$



100% reliable*

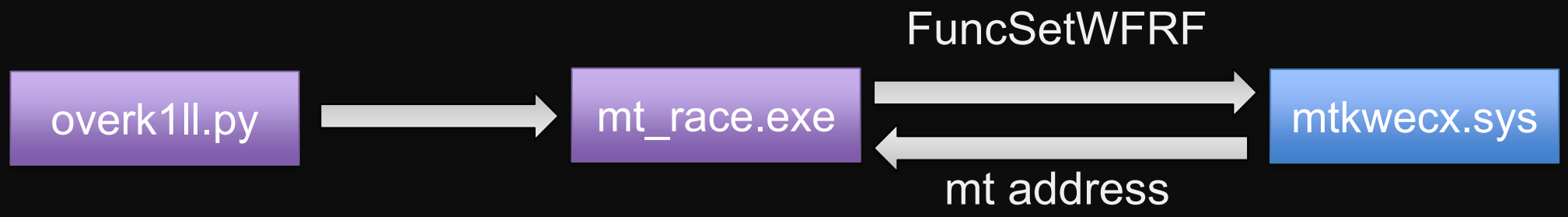
Driver address + prefetch = ❤️

* On a AMD Ryzen 7 5700U

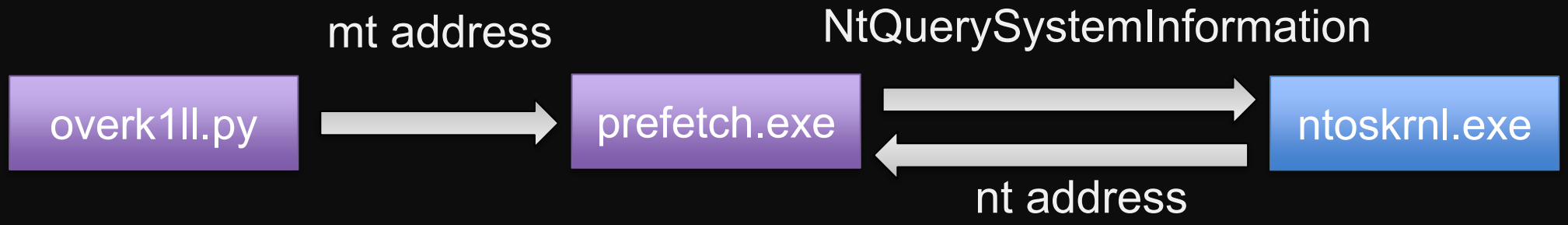
Exploit chain



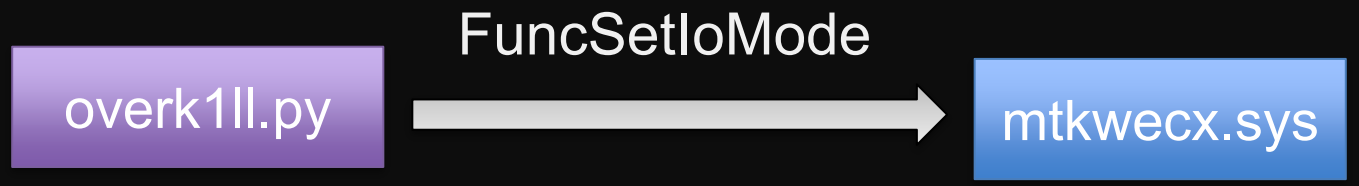
1. Leak mt address (CVE-2025-20740)

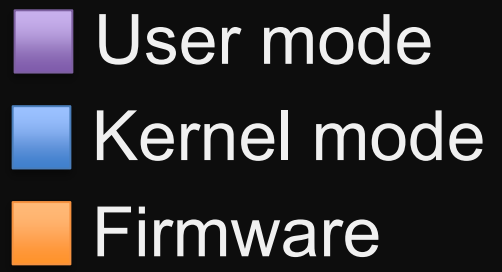


2. Modified prefetch sidechannel

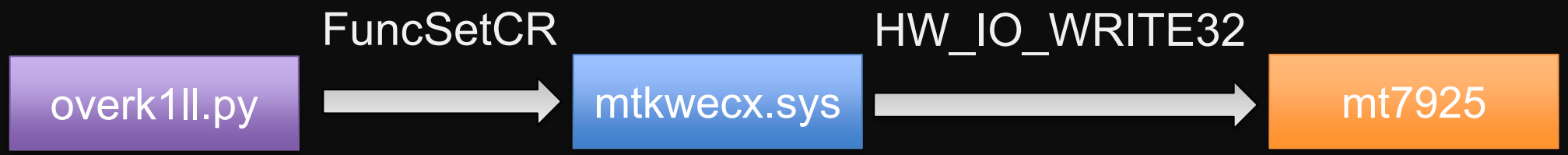


3. Enable MMIO access

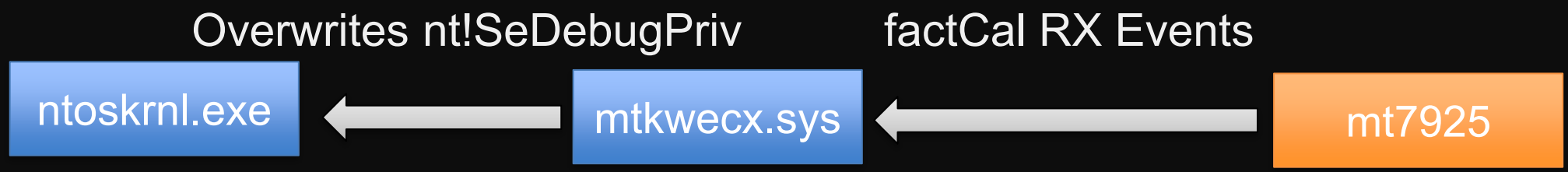




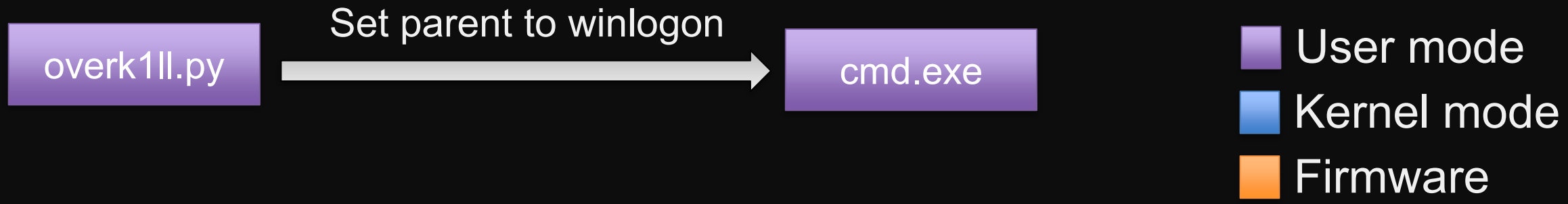
4. Write shellcode to FW (CVE-2026-20423)



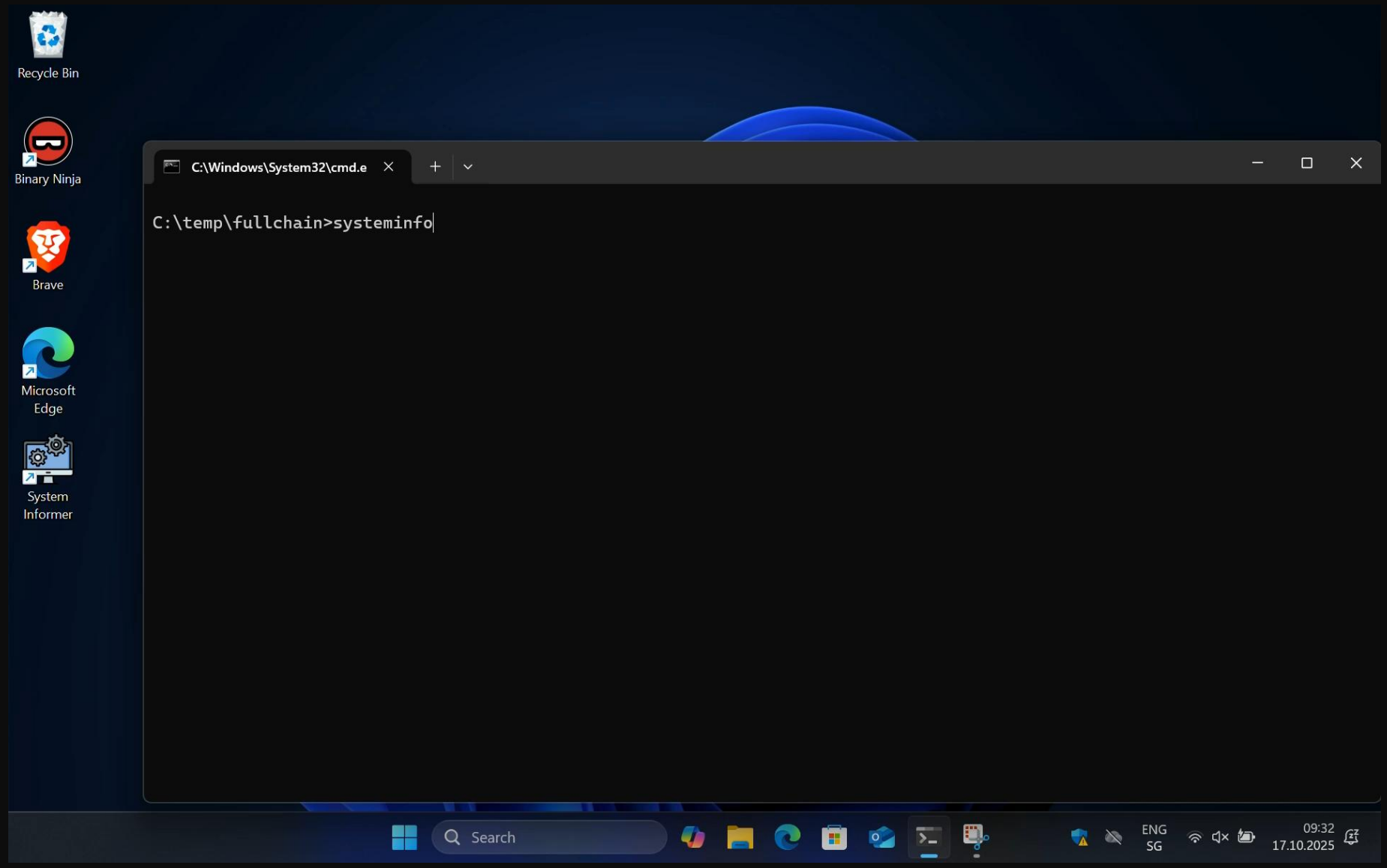
5. FW triggers overwrite (CVE-2026-20436)



6. Spawn SYSTEM shell



Demo


Recycle Bin
Binary Ninja
Brave
Microsoft Edge
System Informer
C:\Windows\System32\cmd.e
C:\temp\fullchain>systeminfo
Search
ENG
SG
09:32
17.10.2025

Patches

CVE-2025-20740	MediaTek Security Bulletin November 2025
CVE-2026-20423 CVE-2026-20436	MediaTek Security Bulletin March 2026

So what does all of this mean?

Contact us?

E-Mail: nicola@vmcall.io

X: @vmcall_io

E-Mail: gueu@zhaw.ch

Blog: https://blog.vmcall.io

Q&A