

# Dancing with Exynos Coprocessor: Pwning Samsung for fun and “profit”

Billy & Ramdhan & Peterpan0927

## About us

- Jheng Bing Jhong, Principal Researcher at STAR Labs
- Muhammad Ramdhan, Principal Researcher at STAR Labs
- Pan Zhenpeng(@peterpan980927), Principal Researcher at STAR Labs

# Agenda

- **Backgrounds**
- Bug analysis
- NPU exploits
- Conclusion

# Android Kernel mitigations

- Android 14 kernel (5.4/5.10/5.15/6.1/6.6)
- PAN/PXN
- UAO
- CFI
- PAC
- MTE
- KASLR
- CONFIG\_INIT\_STACK\_ALL\_ZERO
- CONFIG\_INIT\_ON\_ALLOC\_DEFAULT\_ON
- CONFIG\_DEBUG\_LIST/CONFIG\_SLAB\_FREELIST\_RANDOM/...
- Vendor independent mitigations (KNOX/DEFEX/PhysASLR/...)

# Android exploits

- Universal exploit
- Chipset specific exploit
- Vendor specific exploit
- Model specific exploit

# Android exploits

- Universal exploit
  - Linux kernel bugs: net, binder, etc...
- Chipset specific exploit
- Vendor specific exploit
- Model specific exploit

# Android exploits

- Universal exploit
  - Linux kernel bugs: net, binder, etc...
- Chipset specific exploit
  - Mali GPU, Qualcomm GPU, etc...
- Vendor specific exploit
- Model specific exploit

# Android exploits

- Universal exploit
  - Linux kernel bugs: net, binder, etc...
- Chipset specific exploit
  - Mali GPU, Qualcomm GPU, etc...
- Vendor specific exploit
  - Samsung NPU, Xclipse GPU, Huawei Maleoon GPU, etc...
- Model specific exploit

# Android exploits

- Universal exploit
  - Linux kernel bugs: net, binder, etc...
- Chipset specific exploit
  - Mali GPU, Qualcomm GPU, etc...
- Vendor specific exploit
  - Samsung NPU, Xclipse GPU, Huawei Maleoon GPU, etc...
- Model specific exploit
  - Pixel X driver A, Samsung [A/S/Z] XX driver B, etc...

# Android exploits

- Universal exploit
  - Linux kernel bugs: net, binder, etc...
- Chipset specific exploit
  - Mali GPU, Qualcomm GPU, etc...
- **Vendor specific exploit**
  - **Samsung NPU**, Xclipse GPU, Huawei Maleoon GPU, etc...
- Model specific exploit
  - Pixel X driver A, Samsung [A/S/Z] XX driver B, etc...

# Samsung Exynos Attack Surfaces

- Exynos NPU
- Mali GPU
- Xclipse GPU
- Exynos Baseband

# Why Samsung Exynos?

- Exynos debuted with the Exynos 3110, aiming to build its own SoCs
- Exynos now widely used on EU, ME, India and part of Asia
- In 2022, Xclipse GPU also added to Exynos to replace Mali GPU

# Why Samsung Exynos?

- Exynos NPU and Xclipse GPU both can be accessed from applications
- `allow untrusted_app vendor_npu_device:chr_file { getattr ioctl lock map open read watch watch_reads };`
- `allow untrusted_app gpu_device:chr_file { append getattr ioctl lock map open read watch watch_reads write };`

# Why Samsung Exynos?

- There's not many Exynos bugs or exploits in the past
- NPU/GPU selinux policy makes it perfect targets for LPE
- Samsung is keeping adding new code to it in order to complete its own SoC

# Samsung Exynos Bug hunting

- After 1 month research, we found 13 bugs in Exynos, 10 of which belongs to Exynos NPU, and we will analyze them in the next part

Samsung	NPU	<a href="#">CVE-2025-23095</a>
Samsung	NPU	<a href="#">CVE-2025-23096</a>
Samsung	NPU	<a href="#">CVE-2025-23097</a>
Samsung	NPU	<a href="#">CVE-2025-23098</a>
Samsung	NPU	<a href="#">CVE-2025-23099</a>
Samsung	NPU	<a href="#">CVE-2025-23100</a>
Samsung	NPU	<a href="#">CVE-2025-23101</a>
Samsung	NPU	<a href="#">CVE-2025-23102</a>
Samsung	NPU	<a href="#">CVE-2025-23103</a>
Samsung	GPU	<a href="#">CVE-2025-23104</a>
Samsung	GPU	<a href="#">CVE-2025-23105</a>
Samsung	GPU	<a href="#">CVE-2025-23106</a>
Samsung	NPU	<a href="#">CVE-2025-23107</a>

# Samsung Exynos Kernel Driver

- Main attack surface at `__vertex_ioctl`, reachable by calling `ioctl` on opened `/dev/vertex10`
- `vision_ioctl` -> `vertex_ioctl` -> `__vertex_ioctl`

```
static const struct file_operations vision_fops = {
    .owner = THIS_MODULE,
    .open = vision_open,
    .release = vision_release,
    .unlocked_ioctl = vision_ioctl,
    .compat_ioctl = vision_compat_ioctl32,
    .poll = vision_poll,
    .llseek = no_llseek,
    .flush = vision_flush,
    .mmap = vision_mmap,
};
```

```
static long __vertex_ioctl(struct file *file, unsigned int cmd,
                          unsigned long arg, bool chk_compat)
{
    switch (cmd) {
        case VS4L_VERTEXIOC_S_GRAPH: ...
        case VS4L_VERTEXIOC_S_FORMAT_BUNDLE: ...
        case VS4L_VERTEXIOC_S_PARAM: ...
        case VS4L_VERTEXIOC_S_CTRL: ...
        case VS4L_VERTEXIOC_STREAM_ON: ...
        case VS4L_VERTEXIOC_BOOTUP: ...
        case VS4L_VERTEXIOC_STREAM_OFF: ...
        case VS4L_VERTEXIOC_QBUF_BUNDLE: ...
        case VS4L_VERTEXIOC_QDBUF_BUNDLE: ...
        case VS4L_VERTEXIOC_PREPARE_BUNDLE: ...
        case VS4L_VERTEXIOC_UNPREPARE_BUNDLE: ...
        case VS4L_VERTEXIOC_SCHED_PARAM: ...
        case VS4L_VERTEXIOC_G_MAXFREQ: ...
        case VS4L_VERTEXIOC_PROFILE_READY: ...
        case VS4L_VERTEXIOC_PROFILE_ON: ...
        case VS4L_VERTEXIOC_PROFILE_OFF: ...
        case VS4L_VERTEXIOC_VERSION: ...
        #if IS_ENABLED(CONFIG_NPU_USE_FENCE_SYNC)
        case VS4L_VERTEXIOC_SYNC: ...
        #endif
        case VS4L_VERTEXIOC_QBUF_BUNDLE_CANCEL: ...
        default:
            npu_err("ioctl(%u) is not supported(usr arg: %lx)\n",

```

# Samsung Exynos Kernel Driver

- By opening the driver, it will allocate a new session for us, and it will set `&session->vctx` (`npu_vertex_ctx`) to our `file->private_data`
- `vision_open` -> `npu_vertex_open`

```

struct npu_vertex *vertex = dev_get_drvdata(&vision_devdata(file)->dev);
struct npu_device *device = container_of(vertex, struct npu_device, vertex);
struct npu_vertex_ctx *vctx = NULL;
struct npu_session *session = NULL;
ret = npu_session_open(&session, &device->sessionmgr, &device->system.memory);
if (ret) {
    npu_err("fail(%d) in npu_graph_create", ret);
    _vref_put(&vertex->open_cnt);
    goto ErrorExit;
}
/* set max npu core for the SOC */
session->sched_param.max_npu_core = device->system.max_npu_core;
vctx = &session->vctx;
vctx->id = session->uid;
vctx->vertex = vertex;
mutex_init(&vctx->lock);

ret += npu_queue_open(vctx->queue.inqueue);
ret += npu_queue_open(vctx->queue.otqueue);
file->private_data = vctx;
vctx->state |= BIT(NPU_VERTEX_OPEN);

```

# Samsung Exynos Kernel Driver

- This is what our private\_data holds:

```
struct npu_vertex_ctx {  
    u32 state;  
    u32 id;  
    struct mutex lock;  
    struct npu_queue queue;  
    struct npu_vertex *vertex;  
};
```

# Samsung Exynos Kernel Driver

- This is what our private\_data holds:
- There are 6 states our context can be, it change by reaching specific ioctl
- Also some ioctls require the context to be in the specific state

```
enum npu_vertex_state {
    NPU_VERTEX_OPEN,
    NPU_VERTEX_POWER,
    NPU_VERTEX_GRAPH,
    NPU_VERTEX_FORMAT,
    NPU_VERTEX_STREAMOFF,
    NPU_VERTEX_CLOSE
};
```

```
struct npu_vertex_ctx {
    u32 state;
    u32 id;
    struct mutex lock;
    struct npu_queue queue;
    struct npu_vertex *vertex;
};
```

# Samsung Exynos Kernel Driver

- This is what our private\_data holds:
- npu\_queue contain input/output buffer for interact to the NPU

```

struct npu_queue {
    size_t  insize[NPU_FRAME_MAX_CONTAINERLIST];
    size_t  otsize[NPU_FRAME_MAX_CONTAINERLIST];
    struct npu_queue_list  inqueue[NPU_MAX_QUEUE];
    struct npu_queue_list  otqueue[NPU_MAX_QUEUE];
};
  
```

```

struct npu_vertex_ctx {
    u32 state;
    u32 id;
    struct mutex lock;
    struct npu_queue queue;
    struct npu_vertex *vertex;
};
  
```

# Agenda

- Backgrounds
- **Bug analysis**
- NPU exploits
- Conclusion

# Bug analysis - CVE-2025-23095: Double free

- In `__prepare_IMB_info`, `IMB_info` was allocated and assigned to session first
- If `session->IMB_size` is quite big, it will goto `p_err` and free the `IMB_info` but do not clean the pointer in session
- Reachable via `__vertex_ioctl(VS4L_VERTEXIOC_S_FORMAT_BUNDLE)`

```

if (unlikely(!IMB_info)) {
    npu_err("IMB: IMB_info alloc is fail\n");
    ret = -ENOMEM;
    goto p_err;
}

session->IMB_info = IMB_info;
//...

#if IS_ENABLED(CONFIG_NPU_USE_IMB_ALLOCATOR)
if (session->IMB_size > (NPU_IMB_CHUNK_SIZE * NPU_IMB_CHUNK_MAX_NUM)) {
    npu_uerr("IMB_size(%zu) is larger than %u\n",
            session, session->IMB_size, NPU_IMB_CHUNK_SIZE * NPU_IMB_CHUNK_MAX_NUM);
    ret = -ENOMEM;
    goto p_err;
}
#endif

return ret;

p_err:
if (likely(IMB_info))
    kfree(IMB_info);

```

# Bug analysis - CVE-2025-23095: Double free

- But this IMB\_info could be freed in another function called `_undo_s_graph_each_state`
- At `imb_ion_unmap` routine, it fetched freed `IMB->info` from session and freed it again

```

imb_ion_unmap:
    addr_info = session->IMB_info;
    session->IMB_info = NULL;
    if (likely(addr_info))
        kfree(addr_info);

    __release_imb_mem_buf(session);
#endif
iofm_ion_unmap:
    ion_mem_buf = session->IOFM_mem_buf;
    session->IOFM_mem_buf = NULL;
    if (likely(ion_mem_buf)) {
        npu_memory_free(memory, ion_mem_buf);
        kfree(ion_mem_buf);
    }

```

# Bug analysis - CVE-2025-23095: Double free

- Patch by not free IMB\_info in p\_err code path

```

p_err:
    if (likely(IMB_mem_buf)) {
        kfree(IMB_mem_buf);
        IMB_mem_buf = NULL;
    }
    return ret;
}

```

## Bug analysis - CVE-2025-23097: OOB write

- In function `__vertex_ioctl`, the `npu_profile` array will be updated for every `ioctl`
- The type index is userspace passed `cmd & 0xff`, ranging from 0 to 255

```

struct npu_profile *npu_profile = vctx->profile;
type = cmd & 0xff;
now = ktime_to_us(ktime_get_boottime());
...
temp = ktime_to_us(ktime_get_boottime()) - now;
npu_profile[type].duration += temp;
npu_profile[type].type = type;
npu_log_ioctl_set_date(cmd, 1);

```

# Bug analysis - CVE-2025-23097: OOB write

- But this npu\_profile array only has 20 items
- If we pass bigger cmds it will cause OOB access and corrupt npu\_session

```

struct npu_vertex_ctx {
    u32 state;
    u32 id;
    struct mutex lock;
    struct npu_queue queue;
    struct npu_vertex *vertex;
    struct npu_profile profile[20]; // size 20
};
struct npu_session {
    ...
    void *memory;
    void *exynos;
    struct npu_vertex_ctx vctx; // vctx allocated here
    const struct npu_session_ops *gops;
    int (*undo_cb)(struct npu_session *);

    unsigned long ss_state;
    u32 hids;
    u32 IFM_cnt;
}
    
```

# Bug analysis - CVE-2025-23097: OOB write

- Patch by check type before access npu\_profile

```
if (type < PROFILE_ARRAY_SIZE) {  
    temp = ktime_to_us(ktime_get_boottime()) - now;  
    npu_profile[type].duration += temp;  
    npu_profile[type].type = type;  
}
```

# Bug analysis - CVE-2025-23098: Race double free

- In function `__vb_unmap_dmabuf`, `buffer->dma_buf` is not cleared after unmapping

```
static int __vb_unmap_dmabuf(struct vb_queue *q, struct vb_buffer *buffer)
{
    ...
    if (buffer->vaddr)
        npu_dma_buf_vunmap(buffer->dma_buf, buffer->vaddr);
    if (buffer->daddr)
        vision_dma_buf_dva_unmap(buffer);
    if (buffer->sgt)
        dma_buf_unmap_attachment(
            buffer->attachment, buffer->sgt, DMA_BIDIRECTIONAL);
    if (buffer->attachment)
        dma_buf_detach(buffer->dma_buf, buffer->attachment);
    if (buffer->dma_buf)
        dma_buf_put(buffer->dma_buf);

    buffer->attachment = NULL;
    buffer->sgt = NULL;
    buffer->daddr = 0;
    buffer->vaddr = NULL;
    buffer->size = 0;
    ...
}
```

# Bug analysis - CVE-2025-23098: Race double free

- Function `npu_vertex_unprepare` and `npu_vertex_flush` both can reach `__vb_unmap_dmabuf`
- But they use different lock, `npu_vertex_unprepare` use `vctx->lock` while `npu_vertex_flush` use `vctx->vertex->lock`
- Which means they can be raced and put twice on same dmabuf

```

static int npu_vertex_unprepare(struct file *file, struct vs4l_container_list *clist)
{
    int ret = 0;
    struct npu_vertex_ctx *vctx = file->private_data;
    struct npu_queue *queue = &vctx->queue;
    struct mutex *lock = &vctx->lock;
    ...
    if (mutex_lock_interruptible(lock)) {
        npu_ierr("fail in mutex_lock_interruptible\n", vctx);
        return -ERESTARTSYS;
    }
}

static int npu_vertex_flush(struct file *file)
{
    int ret = 0;
    struct npu_vertex_ctx *vctx = file->private_data;
    struct npu_session *session = container_of(
        vctx, struct npu_session, vctx);
    struct npu_vertex *vertex = vctx->vertex;
    struct mutex *lock = &vertex->lock;
    if (fatal_signal_pending(current) || (current->exit_code != 0)) {
        mutex_lock(lock);
    }
    ...
}

```

# Bug analysis - CVE-2025-23098: Race double free

- Patch by NULL it after dma\_buf\_put

```
    dma_buf_detach(buffer->dma_buf, buffer->attachment);  
    if (buffer->dma_buf)  
        dma_buf_put(buffer->dma_buf);  
  
    buffer->attachment = NULL;  
    buffer->dma_buf = NULL;  
    buffer->sgt = NULL;  
    buffer->daddr = 0;  
    buffer->vaddr = NULL;  
    buffer->size = 0;
```

# Bug analysis - CVE-2025-23099: OOB Access

- `__vertex_ioctl(VS4L_VERTEXIOC_PREPARE_BUNDLE)` -> `npu_vertex_prepare` -> `npu_queue_prepare` -> `npu_queue_alloc`
- It will alloc based on `c->count` (attacker provided value)
- `npu_queue_prepare` refuse to alloc if it has `NPU_QUEUE_STATE_ALLOC`

```
int npu_queue_alloc(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    int ret = 0, i = 0;
    struct nq_container *temp_container;
    struct nq_buffer *temp_buffer;
    struct nq_buffer *temp_buffer_pool;

    if (test_bit(NPU_QUEUE_STATE_ALLOC, &queue_list->state))
        return ret;

    temp_container = kzalloc(c->count * sizeof(struct nq_container), GFP_KERNEL);
    if (temp_container == NULL) {
        ret = -ENOMEM;
        return ret;
    }
}
```

# Bug analysis - CVE-2025-23099: OOB Access

- `__vertex_ioctl(VS4L_VERTEXIOC_PREPARE_BUNDLE)` -> `npu_vertex_prepare` -> `npu_queue_prepare`
- It will set `NPU_QUEUE_STATE_ALLOC` to the `queue->state`

```
int npu_queue_prepare(struct npu_queue *queue, struct vs4l_container_bundle *cbundle)
{
    ...
    ret = npu_queue_alloc(inqueue, cbundle->m[0].clist);
    if (ret) {
        npu_err("npu_queue_alloc(in) is fail(%d)\n", ret);
        goto p_err;
    }

    ret = npu_queue_alloc(outqueue, cbundle->m[1].clist);
    if (ret) {
        npu_err("npu_queue_alloc(out) is fail(%d)\n", ret);
        goto p_err;
    }

    set_bit(NPU_QUEUE_STATE_ALLOC, &inqueue->state);
    set_bit(NPU_QUEUE_STATE_ALLOC, &outqueue->state);
}
```

# Bug analysis - CVE-2025-23099: OOB Access

- If we call `npu_queue_alloc`, first with smaller `c->count` it will success and set state `NPU_QUEUE_STATE_ALLOC`
- Call `npu_queue_alloc` second time with larger `c->count` it will not reallocate again
- It will return 0, and proceed to use newest provided larger `c->count`

```

int npu_queue_alloc(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    int ret = 0, i = 0;
    struct nq_container *temp_container;
    struct nq_buffer *temp_buffer;
    struct nq_buffer *temp_buffer_pool;

    if (test_bit(NPU_QUEUE_STATE_ALLOC, &queue_list->state))
        return ret;

    temp_container = kzalloc(c->count * sizeof(struct nq_container), GFP_KERNEL);
    if (temp_container == NULL) {
        ret = -ENOMEM;
        return ret;
    }

```

# Bug analysis - CVE-2025-23099: OOB Access

- Patch by return -EINVAL in npu\_queue\_alloc

```

int npu_queue_alloc(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    int ret = 0, i = 0;
    struct nq_container *temp_container;
    struct nq_buffer *temp_buffer;
    struct nq_buffer *temp_buffer_pool;

    if (test_bit(NPU_QUEUE_STATE_ALLOC, &queue_list->state)) {
        npu_err("inqueue/otqueue is already in alloc state\n");
        ret = -EINVAL;
        return ret;
    }

    temp_container = kzalloc(c->count * sizeof(struct nq_container), GFP_KERNEL);
    if (temp_container == NULL) {
        ret = -ENOMEM;
        return ret;
    }
}

```

# Bug analysis - CVE-2025-23100: NULL Pointer DoS

- For a `npu_session` obj, the `out_fence_fd` array was inited to -1, and in function `npu_session_queue`, it will allocate a new fd and corresponding `npu_sync_file` object

```
#if IS_ENABLED(CONFIG_NPU_USE_FENCE_SYNC)
/* prepare sync timeline for out-fence */
if (test_bit(VS4L_CL_FLAG_ENABLE_FENCE, &incl->flags)) {
    session->out_fence_value++;
    if (session->out_fence_fd[otcl->index] == -1) {
        ret = npu_sync_create_out_fence(session->out_fence, session->out_fence_value);
        if (ret <= 0) {
            npu_uerr("failed in npu_sync_create_out_fence\n", session);
            goto p_err;
        }
    }
}
```

# Bug analysis - CVE-2025-23100: NULL Pointer DoS

- But after `fd_install`, user can replace this `fd` as another file type, and later in another branch , it call function `npu_sync_update_out_fence` with the previously created and stored `fd` in `session->out_fence_fd`
- Function `npu_dma_fence_to_sync_pt` didn't check `fence` is NULL or not before dereference it, leading to NULL pointer DoS

```
int npu_sync_update_out_fence(struct npu_sync_timeline *obj, int value, int fd)
{
    int ret = 0;
    struct npu_sync_file *sync_file;
    struct npu_sync_pt *pt;
    struct dma_fence *fence;

    fence = npu_sync_file_get_fence(fd);
    pt = npu_dma_fence_to_sync_pt(fence);
    sync_file = npu_sync_file_get(fd);
}
```

# Bug analysis - CVE-2025-23100: NPU FW DoS

- Actually this CVE contains two issues, another one is from NPU firmware side
- This one was found by random test, calling VS4L\_VERTEXIOC\_BOOTUP ioctl with different types will make NPU firmware watchdog timeout thus make AP also crash.

```
[ 75.295801] [0:   InputReader: 1262] !@notifyKey(116), action=0
[ 75.296187] [0:   InputReader: 1262] !@interceptKeyBeforeQueueing(116), action=0, interactive=false
[ 75.306478] [0:   InputReader: 1262] !@Screen_On - 1 : wakeUp: (uid: 1000 pid: 1069) (power_button) (11ms) groupId=0
[ 75.312614] [4:sysinput@1.3-se: 723] tsp: [sec_input] cmd_store: push cmd: set_sip_mode,0
[ 75.312630] [4:sysinput@1.3-se: 723] tsp: [sec_input] cmd_store_function: cmd = set_sip_mode param = 0
[ 75.312634] [4:sysinput@1.3-se: 723] tsp [sec_input] power_state:LP, check_power:ON/LP, wait_result:0
[ 75.312642] [4:sysinput@1.3-se: 723] goodix_i2c 4-005d: [sec_input] [GTP-INF][set_sip_mode:1995] sip mode : off
[ 75.315311] [7:   InputReader: 1262] !@handleInterceptActions(116), action=0, wmActions=0
[ 75.320019] [3:android.display: 1116] usb_notify: usb_sl_show secure_lock = 0
[ 75.321115] [5:           id:11863] debug-snapshot dss: Caller: mailbox_init+0x3b0/0x58c [npu], WDTRESET right now!
[ 75.321187] [3:android.display: 1116] usb_notify: set_notify_lock_state unlock host cable=0, restricted=0
[ 75.321233] [3:android.display: 1116] usb_notify: usb_sl_store secure_lock = 0
```

# Bug analysis - CVE-2025-23101: Arb Free

- In `get_vs4l_profiler_node_child`, there's a child array contains address from userspace. In the loop, `copy_from_usr_vs4l` copy data from user to kernel address `tmpkp`, and update child array to this kernel address

```

// NPU & DSP HW node
for (done_cnt = 0; done_cnt < 3; done_cnt++) {
    if (!child[done_cnt]) continue;

    tmpkp = copy_from_usr_vs4l((void __user *) child[done_cnt],
                              sizeof(struct vs4l_profiler_node),
                              "profiler_child");
    if (IS_ERR(tmpkp)) {
        ret = (int) PTR_ERR(tmpkp);
        goto p_err_profiler_child2;
    }
    child[done_cnt] = tmpkp;
}

static inline void* copy_from_usr_vs4l(void __user *up, size_t len, const char *msg)
{
    int ret = 0;
    void *kp = NULL;

    kp = kzalloc(len, GFP_KERNEL);
    if (!kp) {
        npu_err("%s: kzalloc fail\n", msg);
        return ERR_PTR(-ENOMEM);
    }

    ret = __copy_from_usr_vs4l(kp, up, len, msg);
    if (ret) {
        kfree(kp);
        return ERR_PTR(ret);
    }
    return kp;
}

```

# Bug analysis - CVE-2025-23101: Arb Free

- In `get_vs4l_profiler_node_child`, the child array's size is 3, contains address from userspace. In the loop, `copy_from_usr_vs4l` copy data from user to kernel address `tmpkp`, and update child array to this kernel address
- But it only process 2 in a loop, so `child[2]` left with user controlled address

```

static int
get_vs4l_profiler_node_child(struct vs4l_profiler *kp, bool chk_compat)
{
    ...
    ret = __copy_from_usr_vs4l(tmpkp, (void __user *)kp->node->child,
                               usize, "profiler_child");

    if (ret) goto p_err_profiler_child1;

    if (unlikely(chk_compat)) {
        for (i = 0; i < 3; i++)
            child[i] = (struct vs4l_profiler_node *) ((ulong) tmp_child[i]);
    }
    for (done_cnt = 0; done_cnt < 2; done_cnt++) { // [1] only process 2
        if (!child[done_cnt]) continue;

        tmpkp = copy_from_usr_vs4l((void __user *) child[done_cnt],
                                   sizeof(struct vs4l_profiler_node),
                                   "profiler_child");

        if (IS_ERR(tmpkp)) {
            ret = (int) PTR_ERR(tmpkp);
            goto p_err_profiler_child2;
        }
        child[done_cnt] = tmpkp;
    }
    kp->node->child = child;
}
    
```

# Bug analysis - CVE-2025-23101: Arb Free

- With child[2] can be controlled with any address, it will finally reach free\_vs4l\_profiler\_node to free all three items in kn->child(kp->node->child)
- This will give us an arbitrary free primitive, which is quite strong for exploit

```
static inline void free_vs4l_profiler_node(struct vs4l_profiler_node *kn)
{
    int i;

    if (!kn) return;
    if (!kn->child) goto free_kn;

    for (i = 0; i < 3; i++) {
        if (kn->child[i]) kfree(kn->child[i]);
    }
    kfree(kn->child);
free_kn:
    kfree(kn);
}
```

# Bug analysis - CVE-2025-23101: Arb Free

- Patch by process 3 instead of 2 in a loop

```

// NPU & DSP HW node
for (done_cnt = 0; done_cnt < 3; done_cnt++) {
    if (!child[done_cnt]) continue;

    tmpkp = copy_from_usr_vs4l((void __user *) child[done_cnt],
                               sizeof(struct vs4l_profiler_node),
                               "profiler_child");
    if (IS_ERR(tmpkp)) {
        ret = (int) PTR_ERR(tmpkp);
        goto p_err_profiler_child2;
    }
    child[done_cnt] = tmpkp;
}

```

# Bug analysis - CVE-2025-23102: Double free

- In `__vb_queue_stop`, `q->format.formats` is freed without clearing the pointer.

```
static int __vb_queue_stop(struct vb_queue *q, int is_forced)
{
    int ret = 0;
    u32 i;
    struct vb_bundle *bundle;

    __vb_queue_clear(q);

    ...

    kfree(q->format.formats);

    ...
}
```

# Bug analysis - CVE-2025-23102: Double free

- This kfree can be reached by VS4L\_VERTEXIOC\_STREAM\_OFF
- Double free can be triggered by calling VS4L\_VERTEXIOC\_STREAM\_ON + VS4L\_VERTEXIOC\_STREAM\_OFF in sequence twice.

```
static int __vb_queue_stop(struct vb_queue *q, int is_forced)
{
    int ret = 0;
    u32 i;
    struct vb_bundle *bundle;

    __vb_queue_clear(q);

    ...

    kfree(q->format.formats);

    ...
}
```

# Bug analysis - CVE-2025-23102: Double free

- Patch by set NULL after kfree

```
if (q->format.formats) {  
    kfree(q->format.formats);  
    q->format.formats = NULL;  
}
```

# Bug analysis - CVE-2025-23103: OOB Access

- In `npu_queue_update`, `c` is the user provided container list, it uses user provided container list count to process a loop, and didn't check if it's larger than the actual `queue_list->count`

```
static int npu_queue_update(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    int ret = 0, i = 0, j = 0;
    struct vs4l_container *container;
    struct vs4l_buffer *buffer;
    struct nq_container *q_container;
    struct nq_buffer *q_buffer;

    queue_list->id = c->id;
    queue_list->flags = c->flags;

    for (i = 0; i < c->count; ++i) {
        container = &c->containers[i];
        q_container = &queue_list->containers[i];
        q_container->count = container->count;
        switch (container->memory) {
            case VS4L_MEMORY_DMABUF:
                for (j = 0; j < container->count; ++j) {
                    buffer = &container->buffers[j];
                    q_buffer = &q_container->buffers[j];

                    if (q_buffer->roi.offset != buffer->roi.offset)
                        q_buffer->roi.offset = buffer->roi.offset;
                }
            }
    }
}
```

# Bug analysis - CVE-2025-23103: OOB Access

- It takes q\_container that can come from out-of-bounds, and even it assigns q\_container->count without any check. Then OOB/ARB write could happen

```
static int npu_queue_update(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    int ret = 0, i = 0, j = 0;
    struct vs4l_container *container;
    struct vs4l_buffer *buffer;
    struct nq_container *q_container;
    struct nq_buffer *q_buffer;

    queue_list->id = c->id;
    queue_list->flags = c->flags;

    for (i = 0; i < c->count; ++i) {
        container = &c->containers[i];
        q_container = &queue_list->containers[i];
        q_container->count = container->count;
        switch (container->memory) {
            case VS4L_MEMORY_DMABUF:
                for (j = 0; j < container->count; ++j) {
                    buffer = &container->buffers[j];
                    q_buffer = &q_container->buffers[j];

                    if (q_buffer->roi.offset != buffer->roi.offset)
                        q_buffer->roi.offset = buffer->roi.offset;
                }
            }
    }
}
```

# Bug analysis - CVE-2025-23103: OOB Access

- Patch by check c->count is not vary before loop

```

if (queue_list->count != c->count) {
    ret = -EINVAL;
    npu_err("container_list count should not vary than queue_list count\n");
    return ret;
}

for (i = 0; i < c->count; ++i) {
    container = &c->containers[i];
    q_container = &queue_list->containers[i];
    q_container->count = container->count;
    switch (container->memory) {
    case VS4L_MEMORY_DMABUF:
        for (j = 0; j < container->count; ++j) {
            buffer = &container->buffer[j];

```

# Bug analysis - CVE-2025-23107: OOB Access

- In `fill_vs4l_buffer`, it didn't check if user provided array of buffer in container is smaller than the actual size of buffer in `queue_list`
- In the loop it fetched container from `queue_list->containers`
- Then get the `k` from `queue_list->containers[i]->count`
- Here `c` is user provided buffer, it use `queue_list->count` and `k(queue_list->containers[i]->count)` to iterate `c` and assign values

```

/* sync buffers */
for (i = 0; i < queue_list->count; ++i) {
    container = &queue_list->containers[i];
    k = container->count;
    for (j = 0; j < k; ++j)
        c->containers[i].buffers[j].reserved = container->buffers[j].reserved;
}

```

# Bug analysis - CVE-2025-23107: OOB Access

- If `c->count` is smaller than `queue_list->count`
- Or `c->containers[i].count` is smaller than `k(queue_list->containers[i].count)`
- Then it will lead to ARB read or OOB read.

```

/* sync buffers */
for (i = 0; i < queue_list->count; ++i) {
    container = &queue_list->containers[i];
    k = container->count;
    for (j = 0; j < k; ++j)
        c->containers[i].buffers[j].reserved = container->buffers[j].reserved;
}
  
```

# Bug analysis - CVE-2025-23107: OOB Access

- Patch by check c->count is not vary before loop

```

if (queue_list->count != c->count) {
    ret = -EINVAL;
    npu_err("container_list count should not vary than queue_list count\n");
    return ret;
}

/* sync buffers */
for (i = 0; i < queue_list->count; ++i) {
    container = &queue_list->containers[i];
    k = container->count;
    if (k != c->containers[i].count) {
        ret = -EINVAL;
        npu_err("container_list count should not vary than queue_list count\n");
        return ret;
    }

    for (j = 0; j < k; ++j)
        c->containers[i].buffers[j].reserved = container->buffers[j].reserved;
}

```

# Bug analysis Summary

- Overall we discuss 10(11) different bugs in Samsung Exynos NPU now
- Some of them are only exist on specific model of Exynos, such as CVE-2025-23101, only exist on Exynos 1380
- Some of them exist on both higher end and lower end models, such as CVE-2025-23099 for Exynos 1480 and 2400
- Most of the bugs are exploitable, we will pick two interesting one next as example

# Agenda

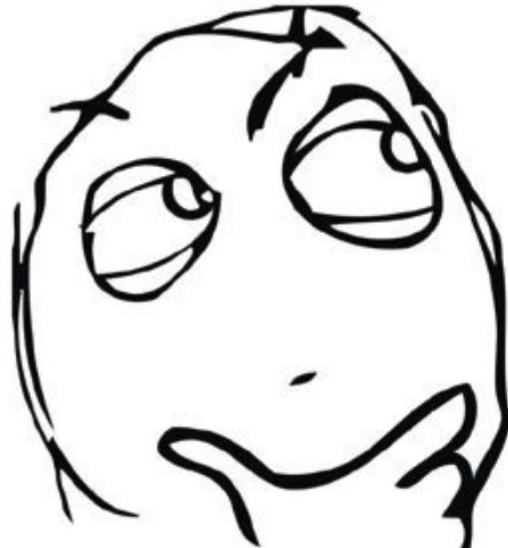
- Backgrounds
- Bug analysis
- **NPU exploits**
- Conclusion

# Bug exploit - CVE-2025-23101

- In case you don't remember this one after 10 bug details, let's do a recap first
- It's an arb free bug in `get_vs4l_profiler_node_child`
- By providing a kernel pointer at `child[2]`, `VS4L_VERTEXIOC_PROFILE_ON` ioctl will free the kernel pointer

# Bug exploit - CVE-2025-23101

- Can we spray a lot and have a fixed address that always has our targeted kernel object?



# Bug exploit - CVE-2025-23101

- Can we spray a lot and have a fixed address that always has our targeted kernel object?
- The answer is yes on Android, by repeatedly rebooting and collect kernel object address (Thanks Magisk), we are able to find a likely address
- And this address could be model/version/memory independent

# Bug exploit - CVE-2025-23101

- In the end we choose a socket buffer to free and reclaim as pipe buffer
- Since during our test it's better to predict this kernel object
- And this object it's easier to reclaim as pipe buffer

# Bug exploit - CVE-2025-23101

- Spray socket buffer
- Call the arb free ioctl to free the socket buffer
- Reclaim socket buffer as pipe buffer
- Read from socket buffer to get pipe's ops
- Use ops to get address of kernel text and selinux\_state
- Spray socket buffer again with selinux\_state address
- Write pipe buffer to disable selinux
- Use pipe r/w to overwrite libc++.so and hijack init to popup reverse shell
- ....

# Bug exploit - CVE-2025-23101

- Besides socket buffer, we can also choose page tables
- Spray a lot of page table entries
- Call arb free ioctl to free the page table pages
- Reclaim as aio pages
- r/w PTE from userspace address
- Samsung Exynos do not have physical memory slide
- Overwrite selinux by linear map address
- Overwrite libc++.so and hijack init to popup reverse shell
- ...

# Bug exploit - CVE-2025-23099

- In NPU Drivers, we can allocate input queue and output queue for communication

# Bug exploit - CVE-2025-23099

- In NPU Drivers, we can allocate input queue and output queue for communication with a limited number of queue (NPU\_MAX\_QUEUE)

```

struct npu_queue {
    size_t  insize[NPU_FRAME_MAX_CONTAINERLIST];
    size_t  otsize[NPU_FRAME_MAX_CONTAINERLIST];
    struct npu_queue_list  inqueue[NPU_MAX_QUEUE];
    struct npu_queue_list  otqueue[NPU_MAX_QUEUE];
};
  
```

```

struct npu_vertex_ctx {
    u32 state;
    u32 id;
    struct mutex lock;
    struct npu_queue queue;
    struct npu_vertex *vertex;
};
  
```

# Bug exploit - CVE-2025-23099

- In NPU Drivers, we can allocate input queue and output queue for communication with a limited number of queue (NPU\_MAX\_QUEUE)
- New allocated queue will be placed at inqueue or outqueue array
- Allocate via `__vertex_ioctl(VS4L_VERTEXIOC_PREPARE_BUNDLE)` -> `npu_queue_prepare`

```

inqueue = &queue->inqueue[cbundle->m[0].clist->index];
otqueue = &queue->otqueue[cbundle->m[1].clist->index];
...
ret = npu_queue_alloc(inqueue, cbundle->m[0].clist);
...
ret = npu_queue_alloc(otqueue, cbundle->m[1].clist);

```

# Bug exploit - CVE-2025-23099

- npu\_alloc\_queue let us to alloc a queue with a numbers of containers, each containers we can allocate a numbers of buffer

```
temp_container = kzalloc(c->count * sizeof(struct nq_container), GFP_KERNEL);
queue_list->containers = temp_container;

for (i = 0; i < c->count; i++) {
    temp_buffer = kzalloc(c->containers[i].count * sizeof(struct nq_buffer), GFP_KERNEL);

    queue_list->containers[i].buffers = temp_buffer;
}
```

# Bug exploit - CVE-2025-23099

- npu\_alloc\_queue will skip if we have allocated queue already

```

int npu_queue_alloc(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    int ret = 0, i = 0;
    struct nq_container *temp_container;
    struct nq_buffer *temp_buffer;
    struct nq_buffer *temp_buffer_pool;

    if (test_bit(NPU_QUEUE_STATE_ALLOC, &queue_list->state))
        return ret;
}
  
```

# Bug exploit - CVE-2025-23099

- npu\_alloc\_queue will skip if we have allocated queue already
- But, it's just return 0

```

int npu_queue_alloc(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    int ret = 0, i = 0;
    struct nq_container *temp_container;
    struct nq_buffer *temp_buffer;
    struct nq_buffer *temp_buffer_pool;

    if (test_bit(NPU_QUEUE_STATE_ALLOC, &queue_list->state))
        return ret;
}
  
```

# Bug exploit - CVE-2025-23099

- npu\_alloc\_queue will skip if we have allocated queue already
- But, it's just return 0, and it continue to call npu\_queue\_mapping

```

ret = npu_queue_alloc(inqueue, cbundle->m[0].clist);
if (ret) { // not reach here, ret = 0
    npu_err("npu_queue_alloc(in) is fail(%d)\n", ret);
    goto p_err;
}
...
set_bit(NPU_QUEUE_STATE_ALLOC, &inqueue->state);

ret = npu_queue_mapping(session->memory, queue->insize, inqueue, cbundle->m[0].clist, is_vmap_req);
if (ret) {
    npu_err("npu_queue_mapping(in) is fail(%d)\n", ret);
    goto p_err;
}

```

# Bug exploit - CVE-2025-23099

- npu\_queue\_mapping will assign new user controlled numbers of container, even if it's larger than previously allocated container!

```

int npu_queue_mapping(struct npu_memory *memory, size_t *size, struct npu_queue_list *
{
    int ret = 0, i = 0, j = 0, k = 0, flag = 0;
    struct vs4l_container *container;
    struct vs4l_buffer *buffer;
    struct nq_container *q_container;
    struct nq_buffer *q_buffer;

    queue_list->id = c->id;
    queue_list->index = c->index;
    queue_list->count = c->count;
    queue_list->flags = c->flags;
    queue_list->direction = c->direction;
  }

```

# Bug exploit - CVE-2025-23099

- npu\_queue\_mapping will assign new user controlled numbers of container, even if it's larger than previously allocated container!
- It will also assign new buffer count on each container

```
queue_list->id = c->id;
queue_list->index = c->index;
queue_list->count = c->count;
queue_list->flags = c->flags;
queue_list->direction = c->direction;

for (i = 0; i < c->count; ++i) {
    container = &c->containers[i];
    q_container = &queue_list->containers[i];
    q_container->count = container->count;
}
```

# Bug exploit - CVE-2025-23099

- npu\_queue\_mapping will assign new user controlled numbers of container, even if it's larger than previously allocated container!
- It will also assign new buffer count on each container
- So we can have malformed number of containers, and number of buffers in every container

```
queue_list->id = c->id;
queue_list->index = c->index;
queue_list->count = c->count;
queue_list->flags = c->flags;
queue_list->direction = c->direction;

for (i = 0; i < c->count; ++i) {
    container = &c->containers[i];
    q_container = &queue_list->containers[i];
    q_container->count = container->count;
}
```

# Bug exploit - CVE-2025-23099

- Primitives #1: Heap OOB read if number of buffer is larger than actual, reachable via `__vertex_ioctl(QBUF_BUNDLE_CANCEL)`

```
static void fill_vs4l_buffer(struct npu_queue_list *queue_list, struct vs4l_container
{
    ...
    /* sync buffers */
    for (i = 0; i < queue_list->count; ++i) {
        container = &queue_list->containers[i];
        k = container->count;
        for (j = 0; j < k; ++j)
            c->containers[i].buffers[j].reserved = container->buffers[j].reserved
    }
}
```

# Bug exploit - CVE-2025-23099

- Primitives #1: Heap OOB read if number of buffer is larger than actual, reachable via `__vertex_ioctl(QBUF_BUNDLE_CANCEL)`
- Primitives #2: If number of container is larger than actual, we can have arbitrary read if we control out-of-bounds container

```

static void fill_vs4l_buffer(struct npu_queue_list *queue_list, struct vs4l_container
{
    ...
    /* sync buffers */
    for (i = 0; i < queue_list->count; ++i) {
        container = &queue_list->containers[i];
        k = container->count;
        for (j = 0; j < k; ++j)
            c->containers[i].buffers[j].reserved = container->buffers[j].reserved;
    }
}

```

# Bug exploit - CVE-2025-23099

- Primitives #3: Heap OOB 4 byte write if number of buffer is larger than actual, reachable via `__vertex_ioctl(QBUF_BUNDLE)`

```

static int npu_queue_update(struct npu_queue_list *queue_list, struct vs4l_container_list *c)
{
    ...
    for (i = 0; i < c->count; ++i) {
        container = &c->containers[i];
        q_container = &queue_list->containers[i];
        q_container->count = container->count;
        switch (container->memory) {
            case VS4L_MEMORY_DMABUF:
                for (j = 0; j < container->count; ++j) {
                    buffer = &container->buffers[i];
                    q_buffer = &q_container->buffers[j];
                    q_buffer->roi.offset = buffer->roi.offset;
                }
                break;
        }
    }
}

```

# Bug exploit - CVE-2025-23099

- Target heap layout

<b>kmalloc-1024</b>
<b>buffers</b>
nq_buffer[0]
nq_buffer[..n]
<b>our_buffers</b>
nq_buffer[0]
nq_buffer[..n]
<b>victim_containers</b>
containers[0].buffers_ptr
containers[..n].buffers_ptr
<b>buffers</b>
nq_buffer[0]
nq_buffer[..n]

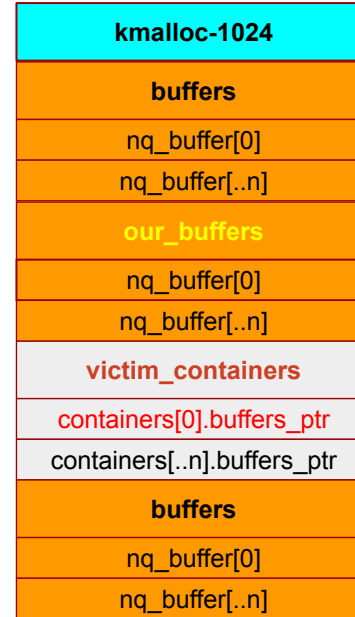
# Bug exploit - CVE-2025-23099

- Target heap layout
- Heap OOB write can write **victim containers[0].buffers\_ptr** from **our\_buffer**

<b>kmalloc-1024</b>
<b>buffers</b>
nq_buffer[0]
nq_buffer[..n]
<b>our_buffers</b>
nq_buffer[0]
nq_buffer[..n]
<b>victim_containers</b>
containers[0].buffers_ptr
containers[..n].buffers_ptr
<b>buffers</b>
nq_buffer[0]
nq_buffer[..n]

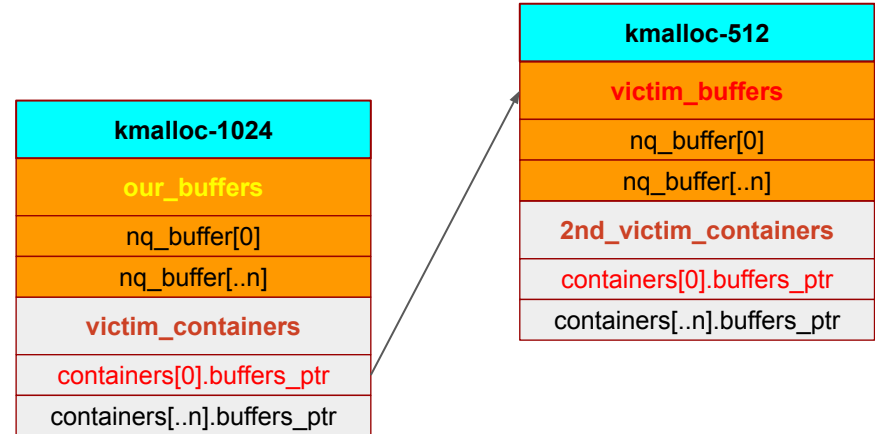
# Bug exploit - CVE-2025-23099

- Target heap layout
- Heap OOB write can write **victim containers[0].buffers\_ptr** from **our\_buffer**
- But it only can write first 4 bytes



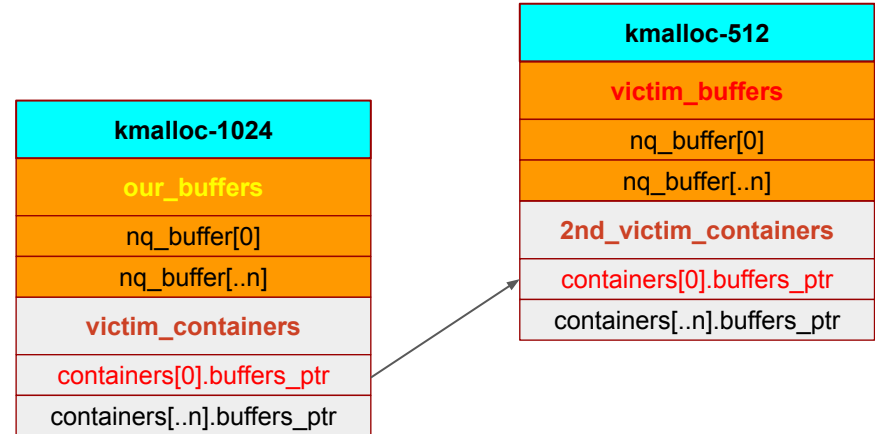
# Bug exploit - CVE-2025-23099

- Target heap layout
- Heap OOB write can write **victim** **containers[0].buffers\_ptr** from **our\_buffer**
- But it only can write first 4 bytes
- Using OOB read we can read **containers[0].buffers\_ptr**



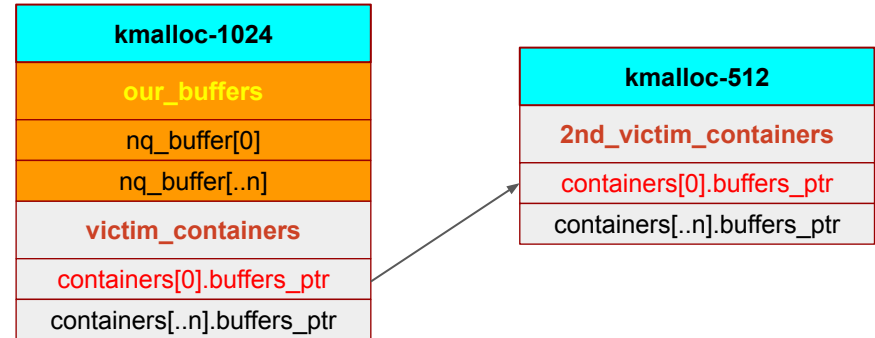
# Bug exploit - CVE-2025-23099

- Target heap layout
- Heap OOB write can write **victim containers[0].buffers\_ptr** from **our\_buffer**
- But it only can write first 4 bytes
- Using OOB read we can read **containers[0].buffers\_ptr**
- Using OOB write 4 byte we change buffers\_ptr to the **second\_victim\_container**



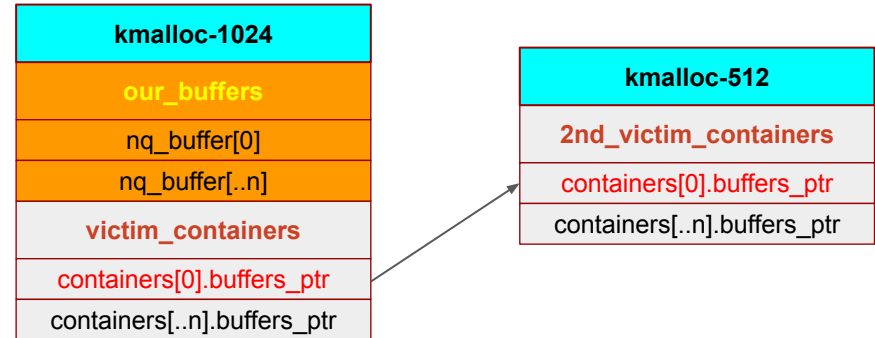
# Bug exploit - CVE-2025-23099

- Using **victim\_containers**, we can modify **second\_victim\_containers[0].buffers\_ptr**
- We still only have 4 byte write
- But we can change **containers[0].buffers\_ptr** to the next 4 byte using **our\_buffers** OOB write
- So we have 8 bytes control on **2nd\_victim\_containers[0].buffers\_ptr**



# Bug exploit - CVE-2025-23099

- So we have 8 bytes control on `2nd_victim_containers[0].buffers_ptr`
- Using the same path code as previous primitives, with controlled `buffers_ptr` we achieve full 8 byte kernel read/write



# Bug exploit - CVE-2025-23099 Demo

```

ffffff8000732000 -> android.wifi.ba
ffffff88a40bcb00 -> pp.smartcapture
ffffff88a40bddc0 -> droid.emergency
ffffff88d74e2580 -> cass
ffffff88a40bb840 -> id.bixby.wakeup
ffffff88d5b88000 -> ng.android.fast
ffffff89aadf4b00 -> .app.aodservice
ffffff8978433840 -> outlineUIProcess
ffffff88da79ddc0 -> kworker/9:5
ffffff8977a7cb00 -> android.vending
ffffff89a7fc4b00 -> ding:background
ffffff89a94312c0 -> android.rkpdapp
ffffff8a3b4512c0 -> sh
ffffff891680ddc0 -> su
ffffff89a8ba5dc0 -> magiskd
ffffff88a5734b00 -> Shutdown thread
ffffff8914905dc0 -> sh
ffffff891678a580 -> npuexp
[+] found current process
fd_array fffffff8025da8000

```

```

$ adb -s R5CX72LPTSA shell
e2s:/ $ getenforce
Permissive
e2s:/ $ nc 127.0.0.1 4444
id
uid=0(root) gid=0(root) groups=0(root),3009(readproc) context=u:r:sec_system_init_shell:s0
getprop ro.build.fingerprint
samsung/e2sxxx/e2s:14/UP1A.231005.007/S926BXXS3AXGD:user/release-keys

```

# Agenda

- Backgrounds
- Bug analysis
- NPU exploits
- **Conclusion**

# Conclusion

- New SoC and drivers are always come with new bugs
- Mitigations need align with all models to take effect
- Fragmented source management is more vulnerable

# Timeline

- Found bugs and write exploits at late 2024
- Report all bugs to Samsung at Dec 2024
- During the fixing process
  - Samsung claimed that all these bugs are not eligible for ISVP because Exynos are not Samsung but Samsung DS (a different company)
  - As Samsung DS is a different company and DS PSIRT operates their own Bug bounty program, Samsung Mobile Security does not and can not handle issues if Exynos bugs
  - In the end Samsung DS reward us 1~3K for each report, depends on affect range and exploit
- Samsung DS fixed all the 13 bugs at June 2025

# References

- [HITCON 2022 - How we use Dirty Pipe to get reverse root shell on Android Emulator and Pixel 6](#)
- [Samsung ISVP Program Rules](#)
- [Black Hat EU 2023: Attacking-NPUs-of-Multiple-Platforms](#)

# Q & A

Thanks for listening