

## Re: DRAM data persistence

---

*Source:* <http://sci.tech-archive.net/Archive/sci.electronics.design/2007-07/msg00552.html>

---

- *From:* Iwo Mergler <[Iwo.Mergler@xxxxxxxxxxxxxxxxxxxxxxxx](mailto:Iwo.Mergler@xxxxxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Thu, 05 Jul 2007 17:57:52 +0100
- 

Nico Coesel wrote:

Jan Panteltje <[pNaonStpealmtje@xxxxxxxxxx](mailto:pNaonStpealmtje@xxxxxxxxxx)> wrote:

On a sunny day (Wed, 04 Jul 2007 17:13:42 GMT) it happened  
nico@xxxxxxxxxxx (Nico Coesel) wrote in  
<[468bd3e8.1017820658@xxxxxxxxxxxxxxxxxxx](mailto:468bd3e8.1017820658@xxxxxxxxxxxxxxxxxxx)>:

You are quoting the C specification. This doesn't mean it is implemented that way it just tells you you must expect rubbish. But like I said, to be absolutely sure no data can be shared between different tasks unintended, any data used by an application is cleared (this does not necessarily mean made 0) by the OS before it is returned to the memory allocation pool.

Well, maybe you should specify *\*what\** OS you are talking about.

Any modern OS should do this.

There seems to be some argument about what happens *\*after\** an application ends.

I only know about Linux, so here is how Linux does it.

After an application ends, absolutely nothing is done with the memory. It gets returned to the kernel pool without clearing.

## Re: DRAM data persistence

The kernel has access to to userspace anyway, so having old application memory contents turn up in the kernel is not a security problem.

To avoid leaking data into other processes, the kernel clears pages as they get mapped into user space.

When a program starts, it gets allocated 3GB (on X86-32) of virtual memory. All this memory is a repeat mapping of a single 4KB page containing zeros, but marked read-only.

If the application now calls malloc, it gets a block of virtual memory containing zeros. Only if the application \*writes\* to the memory, a page fault gets triggered and the kernel maps a memory page at that location.

This memory is cleared before it gets mapped. Here is the kernel source code which does it:

```
/**
 * vmalloc_user – allocate zeroed virtually contiguous memory for userspace
 * @size: allocation size
 *
 * The resulting memory area is zeroed so it can be mapped to userspace
 * without leaking data.
 */
void *vmalloc_user(unsigned long size)
{
    struct vm_struct *area;
    void *ret;

    ret = __vmalloc(size, GFP_KERNEL | __GFP_HIGHMEM | __GFP_ZERO, _KERNEL);
    if (ret) {
        write_lock(&vmlist_lock);
        area = __find_vm_area(ret);
        area->flags |= VM_USERMAP;
        write_unlock(&vmlist_lock);
    }
    return ret;
}
```

Kind regards,

Iwo

.