

# Re: How to develop a random number generation device

---

*Source:* <http://sci.tech-archive.net/Archive/sci.electronics.design/2007-09/msg03384.html>

---

- *From:* Nobody <nobody@xxxxxxxxxxxx>
  - *Date:* Tue, 18 Sep 2007 10:12:07 +0100
- 

On Mon, 17 Sep 2007 23:27:08 +0000, Vladimir Vassilevsky wrote:

With modern hardware (e.g. 80286 and later running in protected mode), the address space of one process (or the OS kernel) simply isn't "visible" to another process.

True, but if you can manage to create a buffer overflow in a kernel process (the TCP/IP stack being a common target here, often implemented as a kernel-level driver), you have the keys to the kingdom.

A messed up data segment is still the data segment. It shouldn't be possible to execute it as a code.

Since 286 there were the goodies like 4 levels of privilege, separate LDTs for every process, different segment rights for code, data and stack. In the theory, that should allow for a pretty solid protection, however in the practice it was (and still is!) unused for the simplicity, sw compatibility and performance reasons.

Agreed.

Some of it is dictated by the language: contrary to what used to be a commonly-held belief amongst DOS programmers, C does not have any concept of "near" and "far" pointers. If you want to use multiple data segments, \*all\* data pointers have to be segment:offset (48 bits on 32-bit CPUs). One data segment (data, bss, rodata, stack) and one code segment wouldn't be a problem, though.

Some of it is dictated by portability: x86 has segmented memory; most other CPUs don't. If you want a single code base to run on multiple architectures, you can't assume segmented memory. This doesn't have much impact upon user space, but the Linux kernel could get quite messy if it

## Re: How to develop a random number generation device

had to allow for disjoint code and data spaces.

OTOH, segmentation doesn't necessarily get you all that much that you don't get with page-level controls (on x86, the inability to map pages write-only is a problem). On newer CPUs, you can implement W^X (write or execute but not both) at page level. On older CPUs, you can put the code first and make the code segment end immediately after the code (all segments must have the same base address to get a single flat address space), but this can cause problems for dynamically-mapped code (dlopen() etc). A compromise is to make the code segment end before the bottom of the stack, which protects against stack-based injection but not the heap or data segment (an attacker would have to find some other vector to get the code called, as you can't trash the return address with a heap overrun).

You could prevent heap overruns if malloc() used a separate segment for every block, but there would be a significant performance hit (malloc() would require a context switch), and you're limited to 8192 (IIRC) local descriptors per process (16 bits for the selector minus 1 bit for global/local and 2 bits for the privilege level leaves 13 bits).

Theoretically you could use the same approach for local (stack) arrays, but the performance hit would be even worse.

.