

Re: How to develop a random number generation device

Source: <http://sci.tech-archive.net/Archive/sci.electronics.design/2007-09/msg02347.html>

- *From:* MooseFET <kensmith@xxxxxxxxxx>
 - *Date:* Wed, 12 Sep 2007 18:11:11 -0700
-

On Sep 12, 2:39 pm, David Brown
<david.br...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

MooseFET wrote:

On Sep 11, 4:58 pm, John Larkin
<jjlar...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:
[... buffer overflow ...]

It sounds to me like C compilers/linkers tend to allocate memory to code, buffers, and stack sort of anywhere they like.

It's up to the linker to build the segments, and the run-time link-loader picks the addresses – the compiler is not involved in the process.

I included the linker in the "compiler/linker" in many cases they are the same program. In many environments the segments end up in memory in the same order as they were in the file.

No the problem isn't really with code mixed with data. It is data mixed with data. The return addresses etc are on the stack along with the local arrays. This means that a routine can overwrite the return address with data by walking off the end of an array. Once that happens the return instruction jumps you to the bad code.

That's a common sort of buffer overflow (it's not the only one, but it's a good example). But as long as you can't execute code in the stack or data segments, it's difficult to exploit such overflows for anything

Re: How to develop a random number generation device

more than crashing the program (since you can only jump to existing code). When the system lets you execute from the stack or the data sections, the attacker has more flexibility since they can insert their own machine code into the program's data, then execute it.

Thus the main protection against malicious buffer overflow attacks is to ensure that the stack and data segments are marked non-executable (something Linux has had for ages, and windows has caught up with – if you enable it). Of course, that means that self-modifying code can't work, which is a problem for just-in-time compilers and certain other techniques such as trampolines.

The other protective mechanism is to randomise the addresses of the segments (supported in Linux, and possibly now in Vista?). This makes it almost impossible to pick suitable addresses when overwriting the return address on a buffer overflow, so that at best you can cause a crash but not specific malicious behaviour.

Why can't at least the compilers be fixed so that they put all the stacks first, then the code, then all the buffers?

Traditionally on most architectures, you get the code first, then the statically allocated data (initialised data followed by uninitialised data), then the heap. The stack starts at the top of memory and grows downwards – the heap and stack meet in the middle when you run out of memory. On modern OS's with virtual memory, the code, data, heap and stack are often separate.

With an x86's MMU, you can make segments for code and stack and the like that have limits on their sizes. The problems can be partly overcome by this.

On most programs, the stack is just above the data segment in physical memory and the malloc() obtained memory is beyond that.