

Re: How to develop a random number generation device

Source: <http://sci.tech-archive.net/Archive/sci.electronics.design/2007-09/msg02435.html>

- *From:* Martin Brown <|||newspam|||@nezumi.demon.co.uk>
 - *Date:* Thu, 13 Sep 2007 07:34:07 -0700
-

On Sep 13, 2:21 pm, MooseFET <kensm...@xxxxxxxxxx> wrote:

On Sep 12, 9:26 pm, JosephKK <joseph_barr...@xxxxxxxxxxxxxxxx> wrote:

John Larkin jjlar...@xx posted to sci.electronics.design:

On Tue, 11 Sep 2007 17:28:32 +0100, Nobody
<nob...@xxxxxxxxxx>
wrote:

On Tue, 11 Sep 2007 07:44:01 -0700, John
Larkin wrote:

Cool. When can we expect
buffer overrun exploits to be
impossible
under Windows?

When it stops letting you run arbitrary
machine code.

Nothing the OS does can prevent machine
code from overrunning a
buffer.

Re: How to develop a random number generation device

Depends on the OS. OS/2 did a pretty good job of preventing anything more than the faulty thread from going down – and that was in the bad old 386 days. IBM hobbled its capabilities by insisting that it ran on 286s too (a fatal error). Intel x86 CPUs onwards had a (largely unused) Bound instruction ever since the iAPX186. You may not be able to stop all buffer overruns, but you can still guard against a lot of them. Almost nobody did :(

Some compilers of the 80's had builtin optional stack and array bounds checking that was very useful during debugging but usually switched off for production builds.

Ancient computers, PDP-11 and VAX certainly, had memory management hardware that separated I and D space, where I space was read-only, and D space could not be executed. And the OS's enforced those rules. It was common to have many users running the exact same code, but mapped into different data spaces.

Problem is, neither Intel nor Microsoft was in the mainstream of computing when they kluged up x86 and Windows.

John

The hardware only became capable of the basics of worthwhile implementation in early Pentiums, and became capable of really worthwhile implementations with Opteron (AMD) and EMT64 (Intel).

386 could do enough in protected mode to make exploits very difficult with the right OS. OS/2 on a 386 was very robust, although its acceptance was stymied by the sheer stupidity of IBM's marketing department.

What nobody at the time could understand was why users loved their Windoze even though it fell over hourly.

I disagree with what you may not have meant to say above. In the

Re: How to develop a random number generation device

microprocessor area, you are largely correct but in other machines, there were many hardware systems that could protect against buffer overflows getting evil code to run. Some of them used a different stack for call and return than for the data. Some such as the IBM-360 didn't have a stack and required each routine to handle its "save area".

Sadly that isn't a good example. It was quite possible to get into Key0 on the old 360 and 370 series machines if you read the right manuals.

Some of the more DSPish machines would also be hard to make a buffers overflow do anything evil. They are far from general purpose machines so although they may show that it could have been done, we can say that they could have made a general purpose PC that was well defended.

It is impossible to execute data on a Harvard architecture machine. That is one hard line solution.

Simulating that model by aggressively separating readonly code and data segments and only permitting genuine code segments to execute goes a long way towards making things secure. Self modifying code not allowed.

Regards,
Martin Brown

.