RIK FARROW

musings

rik@usenix.org

WELCOME TO THE EIGHTH SECURITY edition of *;login:*. I was asked if I would like to edit one issue of *;login:* per year back in 1998, and the only thing that has changed this year is that I am now editing regular issues of *;login:* as well.

Well, perhaps that's not the only thing that has changed this year. Google has announced that it is partnering with NASA. PalmOne has dropped its proprietary PalmOS and aligned itself with Microsoft. Solaris 11 and FreeBSD 6 are out in Beta. Microsoft and Intel are attempting to finesse the issue of the next DVD format by choosing to back HD DVD as opposed to Sony's Blue Ray.

On the security front, things have been relatively quiet. There was a lot of foo-for-all when Michael Lynn decided to resign from ISS rather than remain silent about the exploit he had written for Cisco IOS. While Lynn delivered his exposé at Black Hat Las Vegas, Cisco and ISS got restraining orders against Black Hat in an attempt to prevent copies of Lynn's presentation from escaping "into the wild." Of course, that attempt failed, and resulted in even more copies of the presentation being passed around. Nothing like screaming "fire" to get attention.

There were immediate fears of the ultimate Internet worm that would bring on "a digital Pearl Harbor." But no worm or exploit has emerged . . . so far. I have written in the past about the relative fragility of the routing infrastructure, and a monoculture of routers will certainly not help things. I'll have more to say about IOS, Cisco's Internet Operating System, later.

But on the MS worm front, nothing nearly as exciting as in past years. No Slammer, spreading faster than any worm ever (90% of vulnerable systems in just 10 minutes). No Blaster, leading to fears that Al Qaeda had launched a cyberattack that had led to the FirstEnergy-initiated blackout in August of 2003. Blaster had nothing to do with it and neither did Al Qaeda; downsizing had much more responsibility for the blackout. Not even a new root vulnerability in Sendmail (last in 2003, and I am happy that things have gone well there).

Does this mean we, programmers and sysadmins, can now rest easy? That the problems with program design and architecture have been solved, and that the worms and exploits are now things of the past?

Hardly.

Weakness in Depth

In the world of security, we like to talk about *defense in depth*, having layers of protection. You know, first

you have a packet filter, then a DMZ, with a firewall between the DMZ and the internal networks. There will be IDS both in the DMZ and at critical points on the internal networks, firewalls on critical systems, host-based intrusion detection, centralized patch management and authentication. Boy, this sounds good just writing about it.

And will it work? What exactly do I mean by "weakness in depth"?

Weakness in depth alludes to the design of the operating systems we use every day. Way back in the dark ages of computing, when a fast processor could execute one million instructions per second and be used to provide winter heat for the building it lived in, deep thinkers came up with the notion of the Trusted Computing Base. The TCB would be the bare minimum amount of code required to execute any application securely. Thus, no matter how poorly and insecurely the code had been written, even if a program was written with malicious intent, it would be impossible to violate the rules set down by the TCB.

Having written that, let's consider what passes for a TCB today. You might think that picking out the TCB from the rest of the OS is a difficult task. But it's not, really, because all computers use similar hardware.

The TCB relies on two hardware features for security: memory management and privilege level. Of these, the privilege level is the simplest. Processors can run in one of two (or more) privilege levels. Some processors have just two; the Intel x86 line has four, but only two are used. It is the highest privilege layer that concerns us. Certain instructions can only be executed when the processor is running at the highest privilege level, usually called Ring 0. And memory management can only be manipulated by code running in Ring 0.

Memory management creates the virtual address space that all processes and the operating system itself execute in. Isolating a process within its own set of pages in memory prevents that process from interfering with others. An aberrant process dies without bringing down the system. At least, that is the theory, one that has worked well on *NIX systems for years, and Windows systems more recently.

Memory management also prevents a process running as a less privileged user from injecting code into a process running with higher privilege. This mechanism has proved to be less than perfect, with recent examples (ptrace in OpenBSD in 2002, mremap in recent Linux kernels). But the concept is sound, even if the implementation has been less than perfect.

If memory management is key, and only code running in Ring 0 can affect it, then it sounds like memory management should make up the bulk of Ring 0. But it doesn't. While memory management is a very important part of Ring 0 code, it is in the minority when it comes to lines of code (LOC). The entire mm directory, which contains not just the memory management code for Linux but other memory-related code, contains 28,000 lines of code in the 2.6.11 kernel, out of a total of millions of LOC.

I can't tell you what fraction of the Windows Server 2003 kernel deals with memory. But I can tell you that most of the blue screens you get in WS 2003 come from faulty device drivers—other code that runs in Ring 0. In both Windows and *NIX, most of the kernel is taken up with device drivers, process scheduling and handling, and the network stacks. There is also code that deals with security, but a lot of this provides support for cryptography, and really doesn't belong in the TCB.

By this time, you can see that the TCB for *NIX and Windows includes the entire kernel and millions of LOC. But that's not all. Both *NIX and Windows run software as privileged users, root in *NIX and LocalSystem in Windows. These users can bypass much of the security imposed by the kernel, even violating the

boundaries created by memory management. So any process that runs privileged gets added to the TCB.

Add to this the shared objects or DLLs that get dynamically loaded into those privileged applications, and what you have is many tens of millions more LOC than appear in the kernels alone.

Security-relevant bugs exist at every one of these multiple layers of code. And that is what I mean by weakness in depth. We have built into our TCB many too many layers, all of which are too complex to be trusted. If you knew just how many layers, you would be astounded.

In Lynn's talk about exploiting Cisco routers, he pointed out how really hard it is to create an IOS worm. Each firmware version uses different addresses, and Cisco IOS has been polished for many years, helping to remove the potential for most buffer overflow and heap pointer exploits.

But Cisco IOS currently runs entirely in Ring 0. It is all kernel, all TCB, and a mistake anywhere compromises the entire system. Running in Ring 0 means there are no performance penalties for context switches—but also less of the built-in security obtained by systems designed to securely isolate non-TCB processes/threads.

During HotOS, I learned about Microsoft's experimental microkernel design, named Singularity (see pp. 80–81 of the October 2005 *;login:*). Singularity, like IOS, runs entirely in Ring 0, avoiding the performance penalties for context switches—Singularity can switch between processes almost two orders of magnitude faster than BSD, which goes through context switching. Again, the penalty is the reduction in security by running all processes in Ring 0.

Alternatives

I have been ranting about the weaknesses in operating systems for many years now (perhaps 13). And in the past several issues of *;login:*, I have requested articles that cover different approaches to security, and I plan on continuing to search for still more approaches.

In this issue, Gernot Heiser writes about L4, a microkernel that focuses on keeping the TCB as small as possible. Heiser and the programmers working on L4 and its derivatives strive to minimize the amount of code that needs to be trusted. Microkernels have come a long way since Mach, so performance should not be the main issue. But will the many layers required on top of a minimal TCB bring about the same issues as seen in the bloated TCBs of today? I hope not.

In the August issue you may have read about Xen. In the Xen approach, a virtual machine monitor, which is much bigger than a microkernel, manages all hardware and presents virtual hardware to complete operating systems. This approach solves the problem of buggy device drivers within operating systems, because those device drivers manipulate virtual devices. It also means that you can run any application supported by that operating system, whereas L4 needs its own layers to supply the system call interface provided by that OS. But Xen, like L4, has limited device support, unlike Windows with its unlimited device support—as long as it is a PC device. And Xen has a much larger TCB than L4.

Other issues of *;login:* have included articles about providing isolation or sandboxing for potentially dangerous applications. This issue contains an article, by Shaya Potter and Jason Nieh, about AutoPod, a mechanism that not only provides for isolation but also allows running servers to be migrated to other systems without stopping the server.

You will also find two articles about firewalls, one that compares firewall performance in two popular open source OSes, and another that uses an OpenBSD-

based firewall running within VMware to protect individual Windows systems (really!). Raven Alder shares her expertise in securing backbone networks, while Chris Jordan provides us with a deeper look at writing good ID signatures. Sam Stover and Matt Dickerson show why you want to create memory dumps when performing forensic examinations, and Thorsten Holz provides a wealth of information about botnets based on his experience with the German Honeynet Project.

Proper configuration goes a long way toward maintaining secure systems, and in this issue Luke Kanies explains why you should be using version control and provides examples of how to use CVS and Subversion in the first of what I hope will be many *;login:* articles about configuration management. Ming Chow writes about his experience teaching a college-level course about security to non-CS majors. And Dan Geer starts off the issue with some interesting numbers and conjectures.

History

This issue includes the final History column by Peter Salus. I want to thank Peter for his many columns, I and encourage you to read this one. But I must say that I don't agree with everything Peter has to say. Unlike Peter, I do believe that the day when our appliances will include network interfaces is not that far away—and here's why.

First, just about everything that has a control system includes a computer. Automobiles have many embedded systems, and luxury models already include a satellite telephone that can disable the car. Bluetooth is becoming common. Researchers are busy designing sensor networks composed of tiny inexpensive computers that learn how to communicate over wireless networks without any form of central management (see the October 2005 *;login:* article by Kandula). If tiny sensor devices designed to work for months on a single AA battery already exist, how can self-configured sensor nets for home appliances be far behind?

The question in my mind today is, Do I want my refrigerator, my stove, and my heat pump running Windows? Or Linux? The TCB in either of these OSes is way too big, way too complicated for something designed to provide a little sensor information and perhaps give me some control. And do I really want to worry about the next worm, be it Windows or Linux, infecting my house? Computer security takes on a whole new meaning when everything is not only computer controlled, but also connected to a network that can reconfigure it. After a virus takes over a house network, will it even unlock the front door so I can get in? Or will it turn on the oven and stovetop, turn off the refrigerator, and turn up the thermostat, all on a scorching summer day?

I seriously believe that microkernels will be playing an important part in our not-so-distant future lives. We need secure embedded systems for our cars, our cell phones, and even our refrigerators. What's currently lacking are the development tools and common API for the myriad devices we will find in our future homes and cars.

Microsoft would love to provide a set of developer tools for embedded systems, and already has a chunk of the embedded market—a chunk that just became larger with the support of PalmOne. As Ross Anderson wrote in *Security Engineering*, Microsoft's success happened because it caters to programmers' needs, not to users'. This is a message that I hope will not be lost on any embedded system designers. The terribly bright people who create these systems often expect that the other programmers writing for these systems will be equally bright and have the same deep level of understanding. They won't. But they *will* want to write in Visual Basic, sigh.