OPINION

Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

Almost all of the articles in this issue have to do with security—not surprising, as this is the annual security issue of *;login:*. Our first article, about protecting open source kernels from exploits hosted in user space, brought to mind something completely different, yet related: the design of the *Titanic*. The *Titanic* was billed as the fastest, safest, most modern ship of its era, but we all know what happened. Hubris, current construction techniques (brittle metals), and bad design all led to her sinking. What I want briefly to focus on are the watertight compartments that were supposed to prevent the *Titanic* from sinking. The watertight compartments were on the lowest level of the *Titanic*, but the barrier walls did *not* reach to the ceiling. As the bow sank, water overtopped the walls, speeding the *Titanic*'s, and 1502 people's, demise.

Compartments

Modern computers also have compartments, arranged via several related mechanisms: memory management, protection rings, and the system trap instruction. Memory management controls the access of regions of memory by user processes. To execute memory in the kernel, the system trap instruction transfers control to kernel code. And while in the kernel, code executes at ring 0, the most privileged ring.

Code executed with ring 0 has access to all memory. This access allows the kernel to read or write from anywhere in user space, so the kernel can copy data to devices or write the results of system calls into the memory of user processes. But executing code in user space while the CPU is in ring 0 is something that never needs to occur, and shouldn't happen. But it can, and does.

In Kemerlis et al., our lead article, the authors describe exploits that involve executing code in user space while still operating in ring 0. That gives the executed code the same access as the kernel itself, allowing an exploit to modify data structures in the kernel, and even to add new code in the form of back doors, keystroke loggers, and network monitors. Like the *Titanic*, the "watertight" compartments in most CPUs don't really work.

Kemerlis et al. have developed a compiler plugin, kGuard, that watches for attempts to jump into code in user space and blocks that from happening. Their method works for both Linux and BSD-based kernels today, and although the default remedy is a kernel panic, it does prevent the exploit from succeeding.

Intel, with its Supervisor Mode Execution Protection (SMEP) appearing in CPUs starting with Ivy Bridge, detects when the CPU attempts to execute code in user

space while the CPU is in ring 0. The operating system has to enable SMEP during booting, and clearing a bit in register CR4 disables it. There are already articles about how to disable this feature in Windows 8 [1, 2], and some information about bypassing it in Linux [3]. Note that PaX [4] really prevents this type of attack, by arranging system memory using segment registers so kernel and user-space memory are truly separated.

Hardware

SMEP provides some hardware support for defending against return-to-user-space exploits. Perhaps if the ability to turn off SMEP could be disabled physically, the current exploit paths would be prevented. SMEP could work like the auditing feature in Linux and BSD kernels, where running `auditctl -e 2` prevents auditing from being changed or disabled without rebooting. I imagine that SMEP must be disabled during the initial boot process, and may remain disabled in some operating systems—ones that don't care about security.

Hardware is, well, hard. Implementing a proposed design change in silicon actually takes years. And once it is present, vendors, such as Intel and AMD, do not want the new feature to turn out to be a mistake. Still, I'd like to see even bigger changes than SMEP in future CPUs, with the ability to have much stronger compartmentalization high on my list.

We already have manycore CPUs, but these CPUs all share the same memory and use the same three hardware security mechanisms we have been using in computers since the 1960s. These hardware features—particularly how memory is managed—do a poor job of supporting microkernels, where only a small kernel runs in ring 0, as in seL4 [5]. Communication within current CPUs is via memory, and moving between regions of memory owned by different processes involves the intervention of the operating system, via both a system trap and a context switch. Of course, processes can decide to share memory regions, but then they need to manage the shared memory themselves. What has been missing is hardware support for message passing.

I'm aware that this is a difficult problem to solve, for many reasons: first, it is not how CPUs have been designed; second, because creating such a design would be revolutionary, it would require all new operating systems and applications. Or would it?

seL4 already can support a Linux API, and MINIX 3 [6] supports POSIX. As for running existing applications on top of very different hardware and operating systems, we have been doing this for years, via virtualization. There are also projects such as Microsoft's Drawbridge, designed to run untrusted applications within their own environment (a picoprocess) while providing as much of the required software stack (Windows libraries and the WinNT API) as needed [7]. Although Drawbridge is designed for today's CPUs, it could run much more securely in hardware-supported compartments. I suspect that open source operating systems could explore similar directions to support existing applications in hardware-supported sandboxes as well.

And even exascale computers (see Knauerhase et al. [8]) could benefit from these changes. In their design, most processors run execution engines, whereas only a few provide operating systems support via control engines. Although I don't know whether this design matches what I have long wanted, it does allow programmers to manipulate *data blocks* as first-class objects, and it might be a huge step toward

an architecture that supports the message passing and strong compartmentalization I keep hoping for.

The Lineup

As mentioned, Kemerlis et al. explain why their solution, kGuard, has become important in protecting systems. Their solution is relatively lightweight (less than 1% performance hit for common server applications) while providing a higher level of security for today's hardware and operating systems.

Antonakakis et al. take us off in a different direction by using DNS error responses to locate bot-infected PCs. Current bots often use algorithmically determined domain names as a method to prevent loss of control of their bots through a takeover of the botnet C&C server. Pleiades watches for unsuccessful DNS resolution requests and filters them using machine-learning algorithms to match bots to known malware, and to identify new versions of malware based on the patterns of domain names generated.

Ceron et al., part of the Brazilian CERT team, have been watching attacks against VoIP servers for over a year. They set up honeypots listening for SIP connections, and recorded what happens after a connection to their fake SIP servers. In their article, they both analyze the traffic they've collected and make suggestions for securing SIP servers.

Winter and Crandall explain how parts of the Great Firewall of China (GFC) function. Through their work with the Tor Project, they have learned a lot about how a state agent goes about censoring the Internet and, in particular, blocking access to both Tor routers and to bridges. Theirs is a fascinating story, as nothing is quite as simple as it might seem.

Shekhar et al. provide a bridge between programming and security. They present AdSplit, a method for hosting advertisements with Android apps that separates them from the applications that would otherwise have included them. For advertisers, AdSplit means that the activity they see has not been spoofed, or the advertising hidden. For users, AdSplit moves advertising into a separate environment, so they don't share the same access privileges/permissions as the apps they are associated with.

Crossing partially over into sysadmin, Ur et al. tell us about their research with passwords and password meters. They have calculated the resistance to cracking various sets of passwords created using different rules. They also tested a variety of measures, using thousands of test subjects, to see what forms of feedback work best in helping users pick good passwords—that is, ones highly resistant to guessing or cracking.

Slipping over into the realm of programming, while not really abandoning security, Jang et al. explain how their Python script, ReDeBug, finds code clones. If you have ever programmed, you have likely borrowed code from other programs and repurposed it. In their research, Jang et al. have found thousands of examples of clones in open source code, but their real focus is on code that has been patched but remains unpatched in places it has been copied to—sometimes for as long as 10 years.

David Blank-Edelman takes us on another Perl adventure, this time into the realm of machine translation. Using a Google app as the example, David shows us how to translate languages and use a (moderately) RESTful Web service using Perl libraries.

Dave Beazley explores Pandas, a Python library for data analysis. Dave shows us how to slice and dice large data structures using both Pandas and numpy, an underlying Python library that provides an array object.

Dave Josephson has discovered a new hammer, and is looking for things to pound. The hammer is Nagios XI, a new interactive interface for Nagios that does so much more than just present an interface for system monitoring. Dave is very pleased.

Robert Ferrell was intrigued by both the Great Firewall of China, and the security design considerations of the typical programmer. He views the GFC from on high, then explains, through a simple analogy, what's wrong with most of our software.

Elizabeth Zwicky has reviewed several books, ranging from homebrew forensics (like CSI in your kitchen), two books supporting new versions of OS X, a book about problem-solving for programmers, and another Scrum book. Hint: she doesn't like all of them. I contribute a review of a short book about LEDs for lighting.

The EVT/WOTE '12 summary from the USENIX Security Symposium appears in this issue as well. Actually, all of the summaries from the Security Symposium appear online. With this issue, summaries will appear online as soon as they have been edited, copyedited, and formatted, and although this does take time, it won't be the several months that we have to wait for the print issue of *;login:*. It also means I can develop more articles for each issue without being so concerned about running out of space.

Unlike the *Titanic*, if your kernel is exploited, you will not even notice it—at first. It won't be a bone-jarring crunch as an iceberg the size of a small town rips through the hull, under the waterline. Instead, a successful exploit will silently provide remote control of your system, from the level with total access. Only failed exploits (that crash your kernel) will be "easy" to notice.

I do hope that some day we will have compartments that are complete, and still provide the same performance that we have today. Who knows? Maybe we can have both better security and more performance with future CPU designs.

To see videos from USENIX Security '12, visit: https://www.usenix.org/conferences/multimedia.

Resources

[1] Windows 8 exploit bypassing SMEP: http://packetstormsecurity.org/files/116618/SMEP_overview_and_partial_bypass_on_Windows_8.pdf.

[2] Bypassing Intel SMEP on Windows 8 x64: http://blog.ptsecurity.com/2012/09/bypassing-intel-smep-on-windows-8-x64.html.

[3] Dan Rosenberg, "SMEP: What Is It, and How to Beat It on Linux": http://vulnfactory.org/blog/2011/06/05/smep-what-is-it-and-how-to-beat-it-on-linux/.

[4] PaX: http://grsecurity.net/.

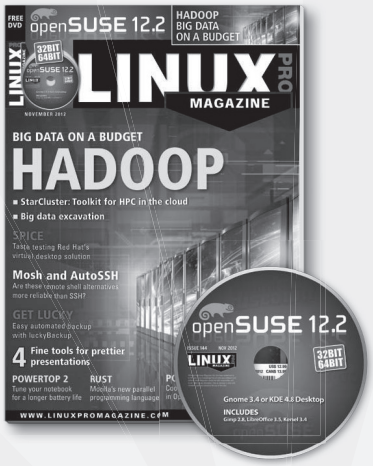
[5] Secure Microkernel Project (seL4): http://www.ertos.nicta.com.au/research/sel4/.

[6] MINIX 3: http://www.minix3.org/.

[7] Drawbridge: http://research.microsoft.com/en-us/projects/drawbridge/.

[8] Knauerhase, Cledat, Teller, "For Extreme Parallelism, Your OS Is Sooooo Last-Millennium," *;login:*, vol. 37, no. 5 (October 2012), USENIX Association.

RISK-FREE TRIAL!



3 issues + 3 DVDs for only $3

PRACTICAL. PROFESSIONAL. ELEGANT.

Enjoy a rich blend of tutorials, reviews, international news, and practical solutions for the technical reader.

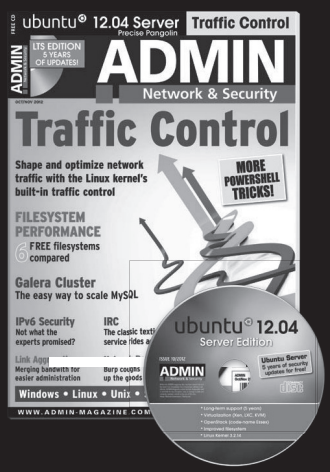
ORDER YOUR TRIAL NOW!

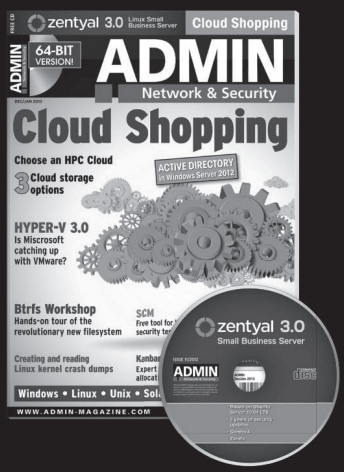
shop.linuxnewmedia.com

2 ISSUES ONLY $15.99

ALSO AVAILABLE

ADMIN: REAL SOLUTIONS FOR REAL NETWORKS





Technical solutions to the real-world problems you face every day.

Learn the latest techniques for better:

- network security
- performance tuning
- system management
- virtualization
- troubleshooting
- cloud computing

on Windows, Linux, Solaris, and popular varieties of Unix.