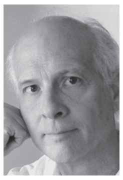
EDITORIAL

Musings

RIK FARROW



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

As often is the case, I find myself musing about the state of computer security. Although there certainly is no easy answer to fixing our insecure systems, I've come across a wonderful analogy, thanks to a NOVA (US public television science) show: "Why Ships Sink" [1].

For ships at sea, as well as airplanes, the answer is often simple: human error is at least partially to blame. But nothing is as simple as it may first appear.

Bulkheads

By the time the *Titanic* sailed, ship designers included bulkheads in their designs. These bulkheads separated the region below the waterline of a ship into separate compartments. The goal for these compartments was to limit flooding if two ships collided. The bow of most ships also included a separate compartment, the *peak tank* that was designed to both crumple and contain any flooding from a collision.

As we all know, *Titanic*'s bulkheads failed rather dramatically. Instead of taking days to sink, *Titanic* took just hours [2]. The bulkheads were not actually watertight but could be, and were, overtopped by flooding. And these compartments were designed under the assumption that a ship would be holed in, at most, two compartments, and then only if a collision occurred right at the boundary between the two compartments.

The sinking of the *Oceanos* [3] provides another vivid example of the failure of watertight bulkheads. Ship designers had done a much better job by this time, having learned from the *Titanic*'s failure. But humans could easily foil this design. In the case of *Oceanos*, partially completed maintenance allowed a leak that started in the engine room to pass through a hole in a bulkhead into the sewage waste disposal tank, and from there, into the rest of the ship via toilets, sinks, and showers. A check valve that would have stopped the backwards passage of water through the waste lines had been removed and not replaced, leading to the sinking of the ship in rough seas off the coast of South Africa.

I certainly find it interesting how both of these examples included assumptions in design and compounded them with the actions of humans.

Computer Security

We have bulkheads, of a sort, in most of our computer systems. Memory management separates access to the memory of one process by other processes. And there are "rings" of protection, with the kernel running in the innermost ring, any VMs and possibly device drivers running in the next one or two levels, and user processes running in the outermost ring [4]. Hardware enforces these rings, so we can imagine them functioning as bulkheads within our computer systems, designed to prevent exploitation, rather than flooding. Attacks at the outermost ring should not impact inner rings.

Like the *Titanic*, which had doors in its bulkheads, CPUs also have "doors" between rings. These provide access to privileged routines—for example, allowing an editor to access blocks on a disk or a Web browser to read data from a network connection. We call these doorways *system calls*, but at the hardware layer they are software interrupts that cause execution to switch between the application that executed the interrupt and the interrupt or trap handler at an inner ring. This handler uses a value to index into the table of system calls.

System calls provide an entry point into the inner rings, and the innermost ring has access to all memory and all hardware. Behind the system call is what provides the weakness in the design: the operating system itself. Operating systems are giant concurrent programs that have several important features: they are crucial for the proper functioning of systems, they are large, and they are difficult to write.

As it turns out, kernel exploits have become one of the most common methods for escalating privilege on *nix systems. In the pre-Internet days, set-user-id (SUID) programs owned by the root were the most popular means for privilege escalation. As the Internet became widely used, root-run network services became more popular targets. And later, largely because of the awareness that both SUID root and root-run network services were dangerous, the numbers of both have decreased over time. There still are many SUID root programs, though not as many as there once were. And the number of root-run network services has declined dramatically. Also, kernel developers have designed kernel-based mechanisms, such as capabilities and SELinux, that can limit the scope of what SUID programs and network services are permitted to do.

That leaves the kernel as a huge program with a complete set of privileges and no limitations. Any code running at ring 0 has complete access to the system, making the kernel a juicy target.

According to conversations with people who run lots of Linux systems, the usual path to exploitation is to gain access to a system through theft of an account, then to use a kernel exploit to gain total control over the system in question. Often, the next step is to install trojan SSH/SSHD programs, so the attacker can steal more accounts.

Our watertight bulkheads are no more watertight, or better designed, than the *Titanic*'s.

The Lineup

We start off this security-focused issue with an article by Jon Howell and friends. Jon and his cohorts have published two papers about Embassies, and after some badgering they have completed an article about their new notion of how Web browsers should work. Instead of building many different browsers that are more like operating systems with lots of leaky bulkheads, they have built a system that runs complete binary applications within a Web browser. Unlike systems such as XaX [5] and Native Client [6] that came before them, Embassies does not require extensive code revisions in applications. Instead, Embassies does something I imagined (and wrote about [7]) many years before. Embassies uses a special library as a replacement for libc and ntdll.dll that provides an extremely limited system call interface to applications. In essence, Embassies reduces the number of openings left in the bulkheads between applications and the kernel to less than ten, far from the hundreds (to thousands) of system calls found today.

Sarah Meiklejohn and her associates wrote about Bitcoin. In their research, they used bitcoins to make online purchases, and through analyzing information used in these transactions, were able to group a goodly fraction of all Bitcoin addresses to a number of well-known entities, such as Mt. Gox and Silk Road. Their work shows that bitcoin transactions are not as anonymous as you might think, and the authors do a great job of explaining both their research and how Bitcoin works.

I interviewed Ben Laurie because a friend had pointed out that he had strong views about Bitcoin. Of course, Ben spends most of his time working to make the Internet safer, through his current work on Certificate Transparency [8]. I did get to ask Ben for his thoughts about digital currencies in general, and Bitcoin in particular.

At Crispin Cowin's request, I asked Istvan Haller and his co-authors to write about their smart fuzzer. Crispin had just been awarded the Test of Time for his work on stack canaries, and he told me this was his favorite work at the 2013 Security conference. Haller et al. combine previous work into a technique that zeroes in on areas within programs that are the best places to find buffer overflows, which is still an issue after all these years.

Although people presented a lot of other exciting research during Security '13, I chose only one other workshop paper for this issue. Mohammad Karami and Damon McCoy had researched DDoS for hire, and presented a workshop paper about this during LEET. I found what they had uncovered fascinating: for a small monthly fee, you can have a service DDoS the IP address of your choice with up to hundreds of millions of packets per second.

Justin Troutman had long been promising me an article about a new framework for cryptography. He and Vincent Rijmen (best known for his part in developing AES) have been researching how best to build a cryptographic framework that works well for both developers and end users. Today, cryptographic APIs leave developers with too many choices to make, choices that should instead be made by cryptographers who understand the theory behind how cryptographic primitives should be used. And most cryptographic libraries result in programs that are difficult for end users to use properly. So instead of having developers making design mistakes to produce programs that end users cannot simply use correctly, Troutman and Rijmen's goal is to create a "green" framework that solves both of these problems.

Phil Pennock contacted me, after some urging by Doug Hughes, about problems he has with how people perceive PGP. Phil runs a keyserver and is a code committer, so he is well situated to comment on PGP. Phil tells us that PGP cannot do many of the things that people expect it to do, for example, prevent traffic analysis. In fact, PGP makes traffic analysis simpler. Phil goes on to explain things you need to know if you want to use PGP properly.

David Lang continues with his series of articles about enterprise logging. David explains how to use the Security Event Correlator, SEC, as a tool for monitoring logs. I had long found SEC a complicated and hard to understand tool, and am glad that David has taken the time to carefully explain how to use some of its most important features.

James Plank has written a survey of erasure codes for storage systems. Most of us are at least somewhat familiar with RAID, a system that in most configurations relies on erasure coding to create a more durable storage system. Jim has presented many papers about erasure coding at FAST workshops, and does a great job in this article of explaining the different ways erasure coding works, and how to measure the effectiveness of erasure coding. Although you might think this is a topic that you don't need to understand, you will understand both erasure coding and RAID much better if you do read his article.

David Blank-Edelman has written about the command line. Sound boring? Well, it's not, as David provides helpful Perl modules and information that makes it easy to parse command lines, and strongly suggests that you not go and build yet another wheel.

David Beazley explains Python packages and what takes the place of main() in Python scripts. I often wondered about this, and, as usual, David provides lots of clear examples of how to access functions within packages as if they were the entry function in a C program.

Dave Josephsen writes to us from the wilderness about his adventures. Well, he only wrote a little bit about hiking in the Rockies, and spent most of his column extolling the usefulness of Go. Dave has discovered that by programming in Go, he has been encouraged to use Git, add network interfaces, think about concurrency, and embrace types and data structures. That's pretty amazing for both a computer language and a curmudgeon like Dave.

Dan Geer and guest co-author Michael Roytman point out that using guesstimates of a vulnerability's likelihood of being exploited makes no sense at all. They share charts and data with us to prove that calculated measures of exploitability do not match up with the vulnerabilities actually exploited, and provide suggestions for doing a better job of deciding what is most vulnerable.

Robert Ferrell gets serious about security in his column. Not that he isn't still being funny, but Robert makes a number of very good points, similar to points I was hearing in hallway talk during the Security Symposium.

Elizabeth Zwicky reviewed five books this time, three on management and two on data analysis. Mark Lamourine reviewed three short books on Vagrant, Git, and Puppet types and providers.

We have many more pages of summaries than we can print, all from the 2013 Security Symposium and the workshops during that week. If you have ever wondered why some things get summarized and others don't, summarizing is a volunteer activity. We do ask any person who has received financial assistance to attend USENIX events to summarize, but we do not force them to summarize. We have learned from experience that the best summaries come from interested participants who have a desire to write summaries.

That said, the volunteers managed to cover all of the Symposium, HotSec, WOOT, LEET, and parts of CSET and HealthTech. We also strive to post the summaries to the *;login:* portion of the USENIX Web site as soon as they have been edited, copyedited, and typeset, and you will often be able to find summaries weeks before they appear in print.

One of the more interesting things I've heard recently about security (which doesn't seem that new at all) is that you don't need to wonder whether your systems will be exploited; you need to *notice* when they have been. If you read the October issue's "For Good Measure" column, you may recall that in the Verizon Data Breach Investigations Report, 80% of data breaches are discovered by some unrelated third party. Geer and Pareek also reported that 65% of the people they survey reported discovering an attack aimed at some other party.

Perhaps we need to quit worrying about the security of our systems, start monitoring for signs of a successful exploit, and keep our incident response teams ready for the emergency that might sink our already leaky ships. We don't have watertight bulkheads, but *Titanic*s cruising serenely along into a night sea scattered with icebergs.

References

[1] Nova on "Why Ships Sink": http://www.pbs.org/wgbh/nova/tech/why-ships-sink.html.

[2] Watertight compartments on the *Titanic:* http://www.titanic-titanic.com/titanic_watertight_compartments.shtml.

[3] Sinking of the *Oceanos:* http://en.wikipedia.org/wiki/MTS_Oceanos.

[4] Rings for computer security: http://arstechnica.com/security/2009/03/storm-over-intel-cpu-security-could-be-tempest-in-a-teapot/.

[5] John R. Douceur, Jeremy Elson, Jon Howell, and Jacob R. Lorch, "Leveraging Legacy Code to Deploy Desktop Applications on the Web," *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI),* December 2008, pp. 339–354.

[6] Native Client: http://code.google.com/p/nativeclient/.

[7] Rik Farrow, "Musings," *;login:*, vol. 32, no. 4, 2007: https://www.usenix.org/publications/login/august-2007-volume-32-number-4/musings.

[8] Certificate Transparency: http://www.certificate-transparency.org/.