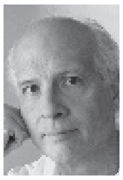
Musings

RIK FARROW



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

I've often dreamed of presenting security as a visualization: images that could clearly convey the dangers represented by different levels of access. My visualization would work so well that even non-technical people would easily understand the relative risks of different attacks. Alas, my skills are lacking when it comes to designing images. But I can write.

Several of the articles found in this issue inspired me in this direction. During my interview with Steve Bellovin, he spoke of walls and gates, nice solid visual metaphors. Sergey Bratus (and others) wrote of the lack of well-defined terms for describing offensive technology, and I certainly agree: the terms we have are often abused and misunderstood. Pete Johnson provided the allegory of a knight being challenged by a gatekeeper before being granted access. No wonder I am thinking in Technicolor.

Beige

Of course, then there's beige, the color of the first IBM PC. These early workstations shared something with their still extant bigger cousins, the mainframes, in terms of access. Rather than a PC, picture a 1970s era mainframe. Got it? Okay, I bet you are visualizing men with pocket protectors and a woman in high heels standing in front of tape drives. The tape drives were much more impressive than the actual mainframes, which were mostly featureless cabinets, often beige or gray. My favorites included lots of blinking lights, including ones attached to memory address lines.

Computer security was equally easy to visualize in that era: physical walls. The mainframe was secured within a special room, and you needed to gain access to that room if you wanted to steal or modify the data, a lot of which was stored on those magnetic tapes. The same was true for PCs for many years, as these were all standalone devices. Not that some mainframes didn't have terminal communication concentrators for remote access, but getting to the data still meant that someone in the secured room would need to heed your request to mount a tape.

The Network

By the end of the '80s, the real era of networking was just beginning. We have to see beyond the walls and locked doors and be able to visualize access to computers in a completely different way. In this case, I always wanted to see something right out of Gibson's *Neuromancer*, where corporate computers were protected by industrial grade "ice": defenses that could, and had, killed intruders. Somehow, Gibson's metaphoric ice was quite visual for me and, I presume, most others who read *Neuromancer*.

But translating ice into something that actually corresponds nicely with the real world of TCP/IP was much more difficult. In that world, what you can *see* from the network are open or closed ports, and the *ice* may or may not be visible as firewalls, and later, intrusion detection systems.

Still, one could have a nice visual representation, in textual form, by using Fyodor's Nmap (nmap.org). As Nmap grew in features and capabilities, you could learn not just which ports were open, but what version of server software was running attached to a port, as well as what operating system supported the software. As real power goes, Nmap still is an incredible tool for visualizing a target textually, but it falls short of Gibson's ice.

We could use (or abuse) Bellovin's walls and gates: Each server is represented by a wall, penetrated by gates that are the open ports. The gates are labeled with the name and version of the service that appears there. I once tried to get a firewall company I was advising to color-code services, from "green" (fairly safe) to "red" (never safe), but they demurred. As this is my vision, I will take another tack at labeling the gates: Ports for insecure services appear as screen doors, while ports for much more secure services look like bank vaults. Too bad that OpenSSL's bank vault door turned out to have a backdoor in it, while appearing quite impressive.

Virtual Walls

Within anything we think of as a computer these days, including smartphones, tablets, desktops, mainframes, and servers within clusters, we also have gates and walls. Steve said, in his interview, that "strong walls are something we're pretty good at ... [but] components have to talk to each other, which implies gates." I've railed for years about the walls we've inherited, since the earliest multiprocessing system designs, and won't go there this time. I will point out that the walls are memory management, used to isolate processes from one another, and various rings of privilege accorded to the operating system by CPU hardware. The most prosaic of these gates are system calls, which allow an unprivileged process to ask the kernel to perform work on the process's behalf. And, as our hardware became more powerful, the number of walls and gates increased as we added virtualization to both hardware and the software that runs on it.

Even here, a bit of visualization might still prove useful. The kernel is like a castle, with a single gate: There is just one way in and one way out, via this gate. Or is there? I'll have more to say about that later, but for now, imagine a castle with an impressive gate. Processes only virtually enter this gate, as the kernel carries out activities vicariously, that is, the proper incantation made at the gate results in the kernel completing some activity and then sharing the results with the process waiting outside the gate. And, while all processes must use the gate, the processes can only interact via the kernel, via the gate of the system call interface.

If you've followed me so far, you are standing outside a castle, among a throng of other busy and eager processes, many clamoring for attention from the gatekeeper. Now that our kernels are multithreaded, it's as if there are many gatekeepers as well, all doing their best to respond to requests so that the processes are not held up. And even if the processes want to communicate with each other, they still must talk to the gatekeeper.

Inside the castle of the kernel, all access is allowed. It is as if the kernel is imbued with a magical quality that provides this level of access—because the kernel has total access. The side effect of this access is that any mistake in the hugely complex kernel can result in sharing this all-powerful access with any evil coder with the right spell: a kernel exploit.

Also, not all processes are treated equally: Even services have their 1%. In the realms of Linux and UNIX, root-owned processes have increased privileges within the castle. In the Windows world, root gets replaced with sets of privileges, mimicking the world of DEC's VMS with both finer control and much more complexity. And although not everyone can be one of the elite, even mere users have resources that exploits can use to abuse or abscond with the user's private data.

Fuzzy Picture

But the castle gate isn't the only way in. I've already mentioned the network, where each open port is like another open gate, each with a completely different set of guards, composed of policy and implementation. Lots can go wrong here, but the main point to keep in mind is that while it might be nice to imagine our castle having only a single entry gate, that's a false image.

And then there are other openings in the wall. In a wonderful presentation, Bill Cheswick described classic castle designs, based on visits he had made to real castles in Europe. But Ches went beyond these descriptions, to the story of the castle that fell because the invaders used a small back door, the one used for convenience by the castle's defenders to visit the town outside.

In my visualization, convenient backdoors look very much like USB ports. Even more than the system call interface, the USB interface is very complicated as it involves both parsing responses to a protocol and running the device driver of the USB device's choice. We all know this attack vector has been used successfully already (Stuxnet), and these convenient backdoors, available to any local attacker, or one that can trick a user into inserting a USB device, make our castle wall look more like Swiss cheese. So much for policy controlled gates.

Personally, I think we need more walls within our castles. At the very least, the gates themselves need to be run within isolated regions, because they too are complicated enough to be exploited.

The Lineup

We begin this issue with an opinion piece by Sergey Bratus, Iván Arce, Michael Locasto, and Stefano Zanero. These men were disturbed by the creation of new laws to regulate the creation, sharing and use of offensive software. Because we have yet to clearly define what exactly we mean by offensive software, new laws, and ones yet to be written, are vague and overreaching. The authors argue for the creation of clearly defined language that will make writing and talking about offensive software, including exploits and vulnerabilities, much clearer and more precise.

I asked Pete Johnson, who had a paper published earlier this year on USB insecurity, to write and explain what's wrong with USB. Pete does a very nice job of explaining how the USB protocol works, as well as how it fails, both through allegory and diagrams.

I'd heard that Stefan Lüders had made presentations about how they handle security at CERN, and I asked him to tell us about that. CERN works with thousands of staff, visitors, and external researchers, which certainly makes security a daunting affair with almost everyone bringing their own device (BYOD). CERN works with people to secure their own devices, as well as educate their owners, but CERN also keeps a stick handy so warnings of failed security cannot be ignored.

Raluca Popa and her co-authors rewrote their NSDI paper on how to secure content on Web servers using encryption. Their solution, in a nutshell, is to handle encryption within the users' Web browsers, moving it away from a Web server that can be subverted or subpoenaed. They have also devised a method that allows searching of stored data on the Web server without sharing keys or using homomorphic encryption.

Chen Chen and his colleagues also rewrote their NSDI paper, and explain how TPM 2.0 can be extended to work through clouds and shared devices. Ordinary TPM can only perform tasks, such as signing a hash or encryption using a stored private key, on the device where TPM is installed. By using a small extension to TPM 2.0, Chen et al. explain how TPM can be leveraged to make sharing encrypted data between devices and clouds work securely.

I decided to interview Steve Bellovin for this issue. Steve has been a figure at USENIX meetings since the UNIX User Group changed its name to USENIX. Steve has also become well known in security through his research, his firewalls book, RFCs, and public speaking. I uncover some of the back story behind many of these accomplishments.

Dilma Da Silva has written an introduction to the Computer Research Association's Committee on the Status of Women in Computing Research (CRA-W) group. CRA-W has done much to help women and minorities succeed in getting into graduate school, publishing, and advancing in their careers. And as Dilma points out, papers with a diverse group of authors tend to get cited more often, implying that the level of creativity and quality is often higher than other paper-writing groups.

Abe Singer volunteered to write about hostbased SSH, a technique he has been using for many years. Although hostbased SSH is not new, it is also often ignored, or at least unknown. Abe explains how hostbased SSH works, why it is better than other techniques, and where it is best used.

Jason Paree writes about event management, a nice term for "handling communications when things go wrong." Instead of the usual way of having too many open lines of communication, which often results in miscommunication and duplicated effort, Paree describes his own group's progress in centralizing communication, documenting, and managing events. For those of you interested in DevOps, event management is an important part of DevOps and getting your process under control.

Andy Seely writes from a manager's perspective about fixing a perception problem: that a part of IT is someone else's problem. Andy actually describes solving a DevOps issue, something I finally recognize after having read *The Phoenix Project* (see my book review). Like the fictional VP of IT in that book, Andy steps in to first understand the problem with one group, get other groups who actually support this group to buy in, and then reorganize to make the changes official.

David Blank-Edelman writes the second of a two-part column about ZeroMQ, a modern message queuing system that simplifies communication between processes, whether on the same system or across a network.

Dave Beazley tackles parsing command line options in Python. Dave begins with a confession, then demonstrates what some of the popular Python modules can do to make parsing options easier.

Dave Josephsen follows a tradition of successful authors who describe the seven habits of successful somethings. Dave, no surprise, explains the seven habits of successful monitoring, starting by telling us that it's about the data, not the tools.

Dan Geer and Joshua Corman take on the myth of the many eyes. The theory has been that open source software should be safer than closed source, but recent discoveries in security-critical open source projects provide fodder for Geer and Corman's investigation.

Robert Ferrell rants about the wonders of various Web tags, including the "Do not track" tag. Along the way, he casts a keen eye on other (current in late May) Internet memes, including Tara the cat, and what it really means when the US indicts five Chinese for stealing IP using the Internet.

Mark Lamourine has tackled a book about understanding the theory of computation. I finally read *The Phoenix Project* and really gained a better understanding of DevOps (and more) from it. I also review a beginner's book on penetration testing that is quite good.

We close out this issue with the summaries from NSDI '14.

I've often visualized computer security in a way not so different from the way I did in these musings. In this alternate scheme, certain programs were red and all the rest were green. If you could trick the red programs into running the code of your choice or accessing resources they were never intended to access, you could imbue your exploit with the color red. The red programs were root-owned processes, set-user-id root programs, and the kernel. Everything else was green by comparison to the power of root, or comparatively privileged parts of Windows.

While we continue to heap praise upon those who manage the feat of separation of privilege (Venema and Bernstein), we keep building monolithic applications with no such separation. Unless we can actually learn how to become designers and programmers who can build carefully limited modules with clear interfaces, we really won't have much use for walls and gates.