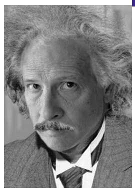
RIK FARROW

musings



rik@usenix.org

I AM NO EINSTEIN, BUT I READ RE-cently that when researchers showed people pictures of smart-looking people, they did better on the multiple-choice test that followed. People shown pictures of bad role models had scores a full 33% lower. In that vein, and with a serious-looking guy in an old suit staring at me, I plan to carry out a thought experiment about the state of computer security.

Like Einstein himself, I will stand on the shoulders of those who have gone before me. Unlike Einstein, most computer security researchers are still alive today, because the field is still young, even if some of us no longer are ungrayed.

First off, I can say with great certainty that the state of computer security is poor. You can see for yourself that it has actually gotten better over time, but even so, you can't browse the Web in safety today. The Provos et al. paper, one of my favorite presentations during the HotBots workshop (see the HotBots summaries later in this issue), points out what Google has started doing as a service to its visitors. Instead of presenting links to Web sites that Google considers harmful, Google instead presents a "Malware Warning" page. If you still want to visit the site you have been warned about, you must cut and paste the link. Provos mentioned that 30–40% of people were clicking through the link on the warning page until Google engineers converted it into plain text. That points to another reason why the Internet will never be safe, but improving good sense in people is not something that can be done with software—at least, not yet.

In six months of searching Google's immense store of cached Web pages, Provos et al. found 450,000 examples of pages that resulted in the compromise of a Windows system when that page was visited using IE. The Google team runs IE and Windows within a VM environment, and it monitors that environment for changes to the registry, new files, and, best yet, new processes. The team also captures downloaded content and scans it for known examples of malware. Based on what its system discovers, a score gets assigned to the page in question, and high-scoring pages get labeled as harmful.

Google's approach appeared a conservative one to me, but when you consider the potential harm of labeling someone's site as harmful, that is a safe approach. I actually searched for examples of harmful pages and discovered, on my very first hit, a nice example of a <script> tag that was being used to download and execute a 67-line Javascript program that checks for a cookie and creates it if it doesn't exist. The script finally creates an <iframe> tag that requests the next stage of drive-by download.

Niels has written a tool that you can use to scan your own Web sites: SpyBye [1]. I suggest that you at least monitor your own Web sites for surprise changes, and for fast-changing sites, SpyBye makes the task much easier to do.

Labeling URLs as harmful is a great symbol for the state of security today. The security industry makes billions of dollars selling tools that search for viruses and attacks, tools that will always be trailing behind the latest attacks. Attackers constantly morph their exploits and malware, making them just different enough to avoid detection. It is much simpler for attackers to generate multiple versions of malware (just change the encryption routine by a few instructions, use a different key, or change the code layout a tad) than it is for A-V or IPS vendors to keep up with this endless stream of new malware.

In my thought experiment, I have seen some nice advances. Techniques such as using nonexecutable stacks (what a great idea!) and randomizing code layout have made buffer-overflow attacks mostly a thing of the past, although not entirely, of course, because we still get patches for various buffer overflows. But these simple changes, even though one of them first required cooperation from some giant CPU vendors, have already make a world of difference. Still, our systems are not secure.

What about the humble Web browser? Our indispensable companion can hardly be considered secure, and there are good, well, bad, reasons for this. First, the Web browser represents a "flat space," as a Microsoft researcher put it during HotOS. What he meant should be obvious with a little thought. Your Web browser runs in a single protection domain, that is, one process. Any event that occurs within that protection domain affects the entire browser. And although browser vendors do attempt to limit the effects of scripts to the page downloaded by a particular site, there are ways around that (see the comments about iframe and script tags made earlier).

Dangerous at Any Speed

And browser technology lends itself to helping attackers. What other software do you use to fetch and execute code, sight unseen, in the context of your own user account? Add to the feature of executing remote code on others' behalf some more exciting features, such as the ability to install plug-ins that run as part of the browser. IE's Browser Helper Objects come to mind, as these are like DLLs and can hook into any event within the browser, for example, keystrokes. I can imagine software that waits until you visit any of a long list of financial institutions, then captures your keystrokes and packages them up neatly in a POST to some offshore hacker haven. But wait! That software already exists, and, according to the Symantex Internet threat summary [2], that offshore haven is the United States!

UK banks have attempted to get around the issue of the theft of authentication details by handing out two factor devices. Such devices are impossible to defeat, as they present pseudo-random passwords for each authentication event. Or are they? I can visualize some browser plug-in that waits until I visit a financial organization, then proxies the communication to an evil server as I authenticate, and finally displays an error to me once authentication is complete. The attacker now has an authenticated connection, via the proxy, to my account, in spite of the use of the authentication device. Bruce Schneier predicted this proxy attack during an RSA conference years ago [3], and malware that does this already exists. Once again, we are defeated by a flat browser space.

I found myself very intrigued by a presentation by a security researcher, Joanna Rutkowska [4], who announced the creation of the Blue Pill, a virtual machine shim that gets installed under Vista at boot time. Now everything above the VM can be controlled, as Vista is just a "guest" operating system! Other security researchers weren't so impressed, because there are too many easier ways to take control of systems. From the browser, down through many layers of GUI libraries, then C libraries, and finally into the kernel itself, the possibilities for exploitation are, well, almost endless. Using the Blue Pill, while interesting, is just overkill on today's systems.

Thought Experiment

Okay, now it's time to do the Einstein thing. Since all I have to do is think, not actually write code or implement hardware, I have total freedom. How can I imagine a way around the security issues we face today?

I think I shall start at the bottom-most level, with a secure boot process. Never mind that someone else has already invented this [5, 6]: my boot process will only load software that has been digitally signed and will use a secure hardware mechanism for verifying this signature. With a secure boot, I can ensure that I won't be swallowing a Blue Pill before I get started!

Next, I conjure up a very small kernel, a microkernel. Large applications are impossible to audit, much less prove that they are secure [7, 8]. By starting small, I will have at least a strong chance that my innermost ring of software can be trusted.

More hardware appears to be an excellent idea, so I will include the concept of hardware-based and -enforced privilege level and memory management. Ignore for the moment that this was invented in the 1960s, because it works poorly when the kernel and other trusted computing base has grown as bloated as it has. The hardware adds to my provably secure kernel because it prevents the kernel code from being modified. The kernel can still modify itself, which I foretell can be a danger, but by sticking to a small kernel I can reduce that threat to a minimum. No one has ever produced perfect software, not even Wietse Venema [9], so I want the added assurance that hardware protection features provide me with.

I seem to recall that early experiences with microkernels were disappointing because they performed poorly when compared to traditional, monolithic kernels. Laying out a traditional kernel design on my garage floor (since no other room is nearly big enough), I notice that, quite unsurprisingly, today's kernel neatly matches up to today's CPU design! Well, that's not much of a surprise, as OS programmers have had 45 years to work with a CPU design that involves using software interrupts for context changes. To say that microkernels perform poorly when compared to monolithic ones is like expecting a championship snowboarder to do well when using a tennis racket as a board. Of course microkernels perform poorly: They are running on hardware that was designed with monolithic kernels in mind!

Now look at that early hardware design. What were they thinking about back in 1960? Those pioneers were wondering how best to go about sharing mainframe computers, because they were extraordinarily expensive to build and maintain. So the early OS designers built time-sharing systems so that many people could use a computer at the same time [10]. This showed real genius, but it makes a poor match for the computing situation we have today. After all, how many people do you share your desktop with when you are using it? There is yourself, along with the botherder who installed bot software and trojans on it, of course.

Today's computers, be they desktops or servers, are essentially single-user systems. There are multiple user accounts, and these are often used to good purpose on server systems. Even desktops have administrative accounts, but this separation of power is usually perverted by allowing the desktop's single regular user administrative privileges. Thus, any exploit against the user has the power of the administrator, proving to be no defense at all. And forget about Vista's new feature that requires entering your password to install any software. Once you have experienced this, well, from what the Microsoft Research guys tell me, you will disable User Account Control, as they told me they quickly did.

And servers? How many users does your Apache Web server run as? How about your Oracle or MySQL server? Just one? I thought so, so we have another case of a single-user system. Even though the server application acts on behalf of perhaps thousands of users, the server itself runs in the context of a single user: just one protection domain. Once again, we are dealing with the aftereffects of time-sharing systems, decades after we switched to new models of computing.

Multicore

You will soon own, if you don't already, multicore CPUs. In fact, I am guessing that if you don't already own these systems, you either manage or work with them routinely. Intel has already demonstrated an 80-core CPU [11], and real 160-core systems are not that far off. Sun has had an 8-core, 32-threaded chip, the T1, for over a year, and Intel and AMD are not far behind.

Multicore processors require new programming techniques to take full advantage of the multiple threads of processing available. We already know how hard it is for human beings to write, debug, and maintain concurrent code (witness the number of years it has taken to get rid of the giant locks in various UNIX-like kernels). And our server and desktop applications are nowhere near to being parallel in design. We will soon have multicore chips that can only be taken advantage of, in a limited way, by the supporting software, the operating system.

In my thought experiment, we have new tools for writing, debugging, and supporting software that deal naturally with parallelism, taking full advantage of having many cores and multiple threads. And I imagine that this may even require some hardware support—for example, for transactional memory (see the summary of David Wood's HotOS keynote on p. 90 of this issue). If parallel programming will require some hardware changes, perhaps it is time to revisit the design decisions that were made in the heat of the Cold War, back in the 1960s.

Sweeping clean my mental workbench, I begin to fashion processor support for fast IPC (Interprocessor Communications). The past CPU designs do this poorly, even though most large applications benefit from sharing memory. Sharing memory is very expensive because it requires that multiple levels of cache, from L1 right up to L3 if it's there, must be kept coherent and consistent. My mental design allows the mapping of small blocks of memory between threads, each running in its own protection domain, without having to invalidate TLB (Translation Lookaside Buffers, used to speed up mapping of virtual memory into physical memory). The old models have got to go if we are ever to support secure microkernels and parallel programming.

I will also pin the microkernel to a single core, with multiple threads but also with L1 and L2 caches large enough that the microkernel runs at or near CPU speeds, not at the much slower speed of RAM.

Finally, I see on my mental workbench my new system: It is totally silent, of low power, and uses 128 threads to swiftly and securely serve my every need. I can browse the Web securely, with content from each site—even images— safely isolated by hardware, in their own protection domains. IPC passes rendered images from one thread to another thread that displays the bitmaps via its device driver, and scripts run only within the context of the site from which they were sourced. This total isolation is made possible through new designs in both processor hardware and software.

Einstein was working as a patent clerk when he wrote his Nobel Prize–winning papers. It was many years before anyone even began to recognize the advances mentioned in his papers written in 1905, and 16 years before he received the Nobel Prize [12].

I surely am not Einstein, even if I have (some) of his hair. But I just as certainly know that we cannot wait 15 years for a change in our computer architecture. We are already way overdue, having lived with time-sharing computer architecture, operating system, and programming paradigms way past their prime. The advent of multicore computing provides us with a rare, and much needed, opportunity to reinvent computing from the ground up.

Lineup

After that rant, you are probably expecting an issue heavy in security and OS design topics. Instead, I have provided some lighter reading for your summer vacation. (The heavier stuff will come later this year, I promise.)

We start off with an article by the authors of the best student paper at NSDI. BitTyrant works like any other BitTorrent client, with one important exception: It rewards reciprocity in a more intelligent manner. Read this article, and learn more about how BitTorrent works and how BitTyrant can improve your download experience.

Alva Couch expounds upon one of the most divisive issues of our time: certification. Rather than traveling down well-worn paths, Alva presents rational ideas, based on real life, as the basis for certifying sysadmins in a practical manner.

Octave Orgeron takes you for a quick tour of Solaris Logical Domains. Logical Domains are a new virtualization technology designed to improve upon existing VM schemes. If anything, Logical Domains appears to me to be a perfect extension to Zones and ZFS. Orgeron plans on writing other articles with tutorials about using it.

Emin Gün Sirer and I next discuss the use and abuse of power laws. Riding on the success of Gordan Moore, we take you on a trip where we extrapolate the future of computing.

Massimo Bernaschi and his co-authors explain their approach for migrating SSL/TLS sockets. Their approach will be appreciated by anyone who wants to improve the reliability of SSL connections with failover servers.

In the columns, David Blank-Edelman starts off with more on unusual but useful Perl modules, focusing this time on tricks with Perl and PDF. Dave

Josephsen joins *;login:* with his introductory column in this issue. Dave's focus is on monitoring and Nagios, but he intends to cover a much broader area than a single Open Source project. I've enjoyed reading Dave's writing in the past, and I think that you, too, will appreciate his insights.

Heison Chak tells us more about branches pulled from Asterisk. Some have not been satisfied with Asterisk and, not being able to pull the code base in their own directions, have branched off. Chak reveals all.

Robert Haskins has taken a sabbatical from "ISP Admin," after many years of writing this column. Many thanks are owed to Bob for writing for *;login:* and helping us see the world of networking from the ISP perspective.

We have, of course, a raft of book reviews, preceded by Robert Ferrell's amusing musings, "/dev/null." Robert bemoans the lack of hat-wearing Texans (oh no!), then lays into the failures of certification. To top off the list, Nick Stoughton tells us where the C standard has been and where the Committee plans to take it next. Nick tells us not to worry, that C will stay the same, yet get better. (Remember what I just wrote about multicore CPUs, and you should appreciate why C must change.)

I'd like to leave you with just two notions from my thought experiment, concepts that are as real as sand at the beach. First, we really stopped using time-sharing systems sometime in the 1980s, and it is time our programming and OS models caught up. Second, microkernels have gotten a bad rap because we ran them on the wrong hardware. Just as you can't jam a DVD into an 8-track player and expect good things to happen, the CPU designs we use must evolve too.

REFERENCES

[1] SpyBye: http://www.monkey.org/~provos/spybye/.

[2] Symantex Internet threat summary: http://www.symantec.com/enterprise/index.jsp.

[3] Bruce Schneier, http://www.schneier.com/blog/archives/2005/03/the_failure_of.html.

[4] Joanna Rutkowska, http://invisiblethings.org/.

[5] Trusted Computing Platform Alliance: http://domino.research.ibm.com/comm/research_projects.nsf/pages/gsal.TCG.html.

[6] Nexus: http://www.cs.cornell.edu/people/egs/nexus/.

[7] Kevin Elphinstone et al., "Towards a Practical, Verified Kernel," http://www.usenix.org/events/hotos07/tech/.

[8] Attacking the Core: Kernel Exploitation Notes: http://phrack.org/issues.html?issue=64&id=6#article.

[9] Wietse Venema, http://www.porcupine.org/wietse/.

[10] MULTICS: http://www.multicians.org.

[11] 80-core demonstration: http://www.engadget.com/2007/02/11/intel-demonstrates-80-core-processor/.

[12] Albert Einstein's biography: http://nobelprize.org/nobel_prizes/physics/laureates/1921/einstein-bio.html.