EDITORIAL

Musings

RIK FARROW

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

I've decided to follow the example of Randall Munroe (xkcd) and work at answering an absurd hypothetical question: Will we ever have secure systems?

Actually, a well-known professor at Purdue, Gene Spafford, already answered this question way back in 1989:

> The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards—and even then I have my doubts. [1]

I've actually used the image of a computer cord coming out of a block of cast cement in some presentations, as there's nothing like a concrete visual image to help people understand the problem.

Input Challenged

Instead of a computer buried in a concrete block, I have a simpler suggestion: Let's have a computer, running any OS you like, but not permit any input to it. If it crashes, the BIOS will be set to reboot the OS, then the computer just goes on sitting there, with the OS sitting in an idle loop.

This doesn't move the state of the art in a much more useful direction than the computer-in-concrete version, but it is suggestive: It's not the computer running the OS that's the problem, it's the input that gets fed to programs running under that OS. And that's the conundrum: If you want a secure computer, don't allow anyone to access it. We still have a useless computer, unless you are using it to heat a room.

To illustrate just how bad the problem can be when you allow input, I remember the first kernel security bug I'd ever heard of. In the UNIX System III or Version 7 kernel, you could get a root shell by running any program and providing a specially crafted argument to the command you were exec'ing. The argument needed to be longer than 5120 bytes, as that was the statically defined length for execve() call arguments, and by overrunning this buffer, you could overwrite the u_area where the owner and group IDs were stored.

That means that:

```
#include<stdio.h>
main()
{
   printf("Hello World");
}
```

was capable of being used to exploit the system.

Even though the "hello world" program doesn't accept any input, the program executing it does, and there's the rub. So it appears that what might seem to be a simple program—on a computer that has no networking beyond UUCP over serial port and on a kernel short enough to have been published in book form [2] several years earlier—can be rooted.

One useful line of research points to input parsers as being to blame for many successful exploits [3]. The reasoning behind this assertion is clear: An input parser more complex than the simplest parser in the Chomsky hierarchy [4] cannot be proven to work as expected. That simplest parser uses regular expressions where you have a choice of parsing from the left or the right end of your input. Anything more complex than that is asking for trouble.

If you have a difficult time visualizing an input parser, just consider almost any shell script that accepts command-line arguments. If you have written, or seen, such a script, then you should know that the switch or if-then-else statements at the beginning of the script act as an input parser, even if it is a simple one. Other input parsers include Web scripting back-end engines such as PHP, Perl, Ruby, Python; SQL query parsers; the shells; and the Web servers themselves.

During an invited talk at USENIX Security 2014 (see the summaries in the back of this issue), Felix Lindner (FX) provided a wonderful example of a parsing bug. The chunk encoding bug first appeared in the Apache Web server in 2003, and then in Nginx in 2013. The code was different in these two programs, but the bug was almost the same.

Absurd Answer

One of the most popular answers to the question "How do we improve security?" involves the use of security software. This software is supposed to protect us from bugs in other software. But this is absurd, as security software is also software, subject to the same problems as other software. Worse yet, security software, whether it's an IPS or a virus scanner, has to parse input using complex rules, making it even more vulnerable. On top of that, security software generally runs with privileges, making that software an even more exciting target.

Perhaps we could wrap the security software inside of some other software to isolate the rest of the system when the security software gets exploited? Sandboxing, another popular security solution, involves relying on yet more software to make the software we have more secure. It's turtles all the way down.

The Lineup

We begin this issue with an article by Rory Ward, with help from Betsy Beyer. Ward describes how Google is moving beyond the notion of having a privileged network, protected by a firewall that is considered secure. Some Google employees have been working on the many moving parts needed to replace this outdated design with something a lot better thought out and, likely, much more secure. I think it is wonderful that Google management has decided to allow some employees to share information like this with the rest of us.

Pawel Dawidek and Mariusz Zaborski bring us up-to-date on Capsicum. Capsicum, which appeared during Security 2010, uses capabilities to control the namespaces that a process has access to. If you read the "Containers" article in the October 2014 issue of *;login:*, you will be familiar with the Linux approach to this problem. Dawidek and Zaborski explain how sandboxing was done before Capsicum, updates to Capsicum, as well as a server program, casperd, that can help with adding Capsicum to applications.

Santiago Torres and Justin Cappos share some work they have been doing to make the storage of password hashes safer. They've created a scheme, using cryptographic shares, that makes cracking password hashes 23 orders of magnitude more difficult, while still taking a tiny amount of time to perform authentication.

I asked Peter Gutmann to write about his own experience with debugging. Peter shares a technique based on failure as an important debugging tool. Not his own failure, but a method for injecting failures so that the failure paths of programs can be rigorously tested. Not that this would have helped with Heartbleed or Shellshock, as the failures in parsing there weren't tested, but Peter's technique will help you better test your code.

Mark Gondree decided to continue the discussion that was begun by a panel on the "Gamification of Security" at the 3GSE workshop. Mark posed questions to all of the panelists, then collected and edited their responses. If you've wondered about gamification, I think you will learn a lot from reading this discussion.

I wanted to interview Dan Farmer. I met Dan almost 25 years ago, and while I would see him during security conferences, I had lots of unanswered questions about his career. Dan would often base his decisions on ethics rather than personal profit or security, and that's had a huge impact on his life.

Robert Ricci and Eric Eide announce CloudLab. While I heard about this during Security, their announcement goes well beyond just security. They, and a much larger team in multiple locations, are building infrastructure for doing cloud research. CloudLab provides barebones systems, VMs, and access to networking so that a wide variety of cloud research projects can have a realistic test environment.

Andy Seely continues his sysadmin management column with stories about keeping up with details. In each story, someone has ignored some aspect of their professional life, even while doing an otherwise exemplary job, and that has gotten each of them in career/job trouble.

David Blank-Edelman explains how to make the best use of two different search interfaces to CPAN, the Perl module site. There are gems hidden away in each of the GUI interfaces, which David reveals.

David Beazley reveals a new Python 3.4 feature, via explaining constants. Constants are an issue in all scripting languages, as they are, uh, not terribly constant. Enums and IntEnums help with this.

Dave Josephsen waxes enthusiastically about collectd, a client-side agent that is useful for collecting the various bits of info you want to monitor.

Dan Geer takes the concept of the stress testing of the largest banks and turns it into a plan for testing your own security preparedness. Stress testing helps you and your organization evaluate just what level of risk you might be facing when the next Internet worm hits.

Robert Ferrell has dug up another rant, this time on security snake oil. While Robert describes this as a "dystopian future," I think it is a scenario that's all too familiar.

Mark Lamourine has written book reviews about functional programming, a difficult book about SDN, and the new Limoncelli, Chalup, and Hogan book about managing clouds. I've written a (much easier) review about the Randall Munroe book *What If? Serious Scientific Answers to Absurd Hypothetical Questions*.

Most of the workshops that accompanied USENIX Security have some summaries covering them, with the exception of HotSec, which by design is not taped or summarized, and EVT/WOTE. Every session in Security itself, and WOOT, are covered in an excellent set of summaries.

Just as I was editing this column, I learned of a new bug in Bash, which is going by the name "Shellshock." By attempting to create a null function in an environment variable, an attacker can execute anything she likes via the shell. This bug appears to be a problem in parsing, when I looked at the patch files [5] for Bash. One could argue equally that this was a mistake in implementation, as null functions shouldn't be evaluated within environment variables, but that's just splitting hairs. The bug does appear to have been in Bash for many years. And Bash parses its input, as you should expect, but limiting Bash to the simplest Chomsky hierarchy parser would also make Bash a wimpy shell.

The lesson of Shellshock is that you should never expose a shell to input that you don't trust. That shells get invoked in a large variety of software, including DHCP clients, just shows how difficult it is for people to write secure software.

I'd like to end this column with another quote:

> I don't think it's an exaggeration to say that cyber defense solutions will serve as the essential basis for human development and economic growth in this century—I think it's happening before our very eyes.
> —Prime Minister Benjamin Netanyahu [6]

While I'd rather not agree, I can see the logic in this statement. If we build software cyberdefense solutions that are themselves software, then we have created a perpetual motion machine that will benefit the purveyors of security software.

Instead, I believe it would make much more sense to produce software tools without the sharp edges that make writing software so dangerous, so insecure. While this has been attempted (consider Java), part of the problem with this approach is that a new programming environment has to encompass everything that a programmer believes he needs to do, simply, quickly, and securely. Then, perhaps, we would have Web servers invoking shells to process request variables, or DHCP clients [7] invoking a shell to configure the client. And this process must include the OS too, as the largest, most complex, software that we run.

References

[1] Gene Spafford quotes: http://spaf.cerias.purdue.edu/quotes.html.

[2] *Lions' Commentary on UNIX 6th Edition, with Source Code* (Peer to Peer Communications, 1996): http://en.wikipedia.org/wiki/Lions'_Commentary_on_UNIX_6th_Edition,_with_Source_Code.

[3] LANGSEC: Language-Theoretic Security: http://www.cs.dartmouth.edu/~sergey/langsec/.

[4] "Chomsky hierarchy," Wikipedia: http://en.wikipedia.org/wiki/Chomsky_hierarchy.

[5] Patches to Bash: http://ftp.gnu.org/pub/gnu/bash/bash-4.3-patches/bash43-025.

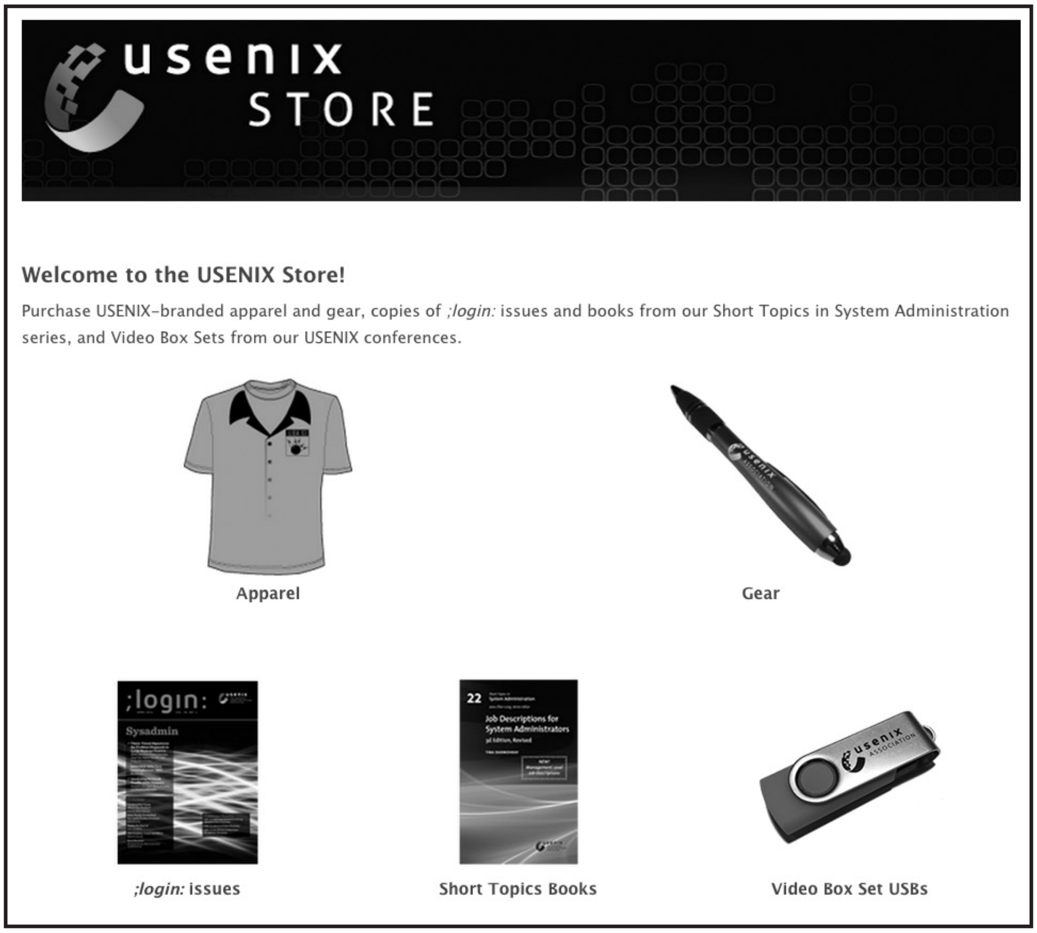
[6] "Netanyahu, Kaspersky, and Gold tackle cyber 'game-changers'," EurekAlert! Press Release for Cyber Week 2014: http://www.eurekalert.org/pub_releases/2014-09/afot-cw2092414.php.

[7] *"ISC's DHCP Client Can Be Used as a Delivery Vector for Bash Bug,"* ISC DHCP, dhcp-4.3.1/client/client.c; http://www.isc.org/.

usenix

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Announcing the USENIX Store!



Welcome to the USENIX Store!

Purchase USENIX-branded apparel and gear, copies of *;login:* issues and books from our Short Topics in System Administration series, and Video Box Sets from our USENIX conferences.

Want to buy a subscription to *;login:*, the latest short topics book, a USENIX or conference shirt, or the box set from last year's workshop? Now you can, via the brand new USENIX Store!

Head over to www.usenix.org/store and check out the collection of t-shirts, video box sets, *;login:* magazines, short topics books, and other USENIX and LISA gear. USENIX and LISA SIG members save, so make sure your membership is up to date.

www.usenix.org/store