EDITORIAL

Musings

RIK FARROW

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

Although *;login:* no longer has theme issues, this issue is loaded with articles about security. Sitting in lofty isolation, I thought I would muse this time about the three problems we have with computer security: hardware, software, and people. I don't think I've left anything out.

Hardware

Most of the computers that people use (as opposed to simpler IoT devices) have hardware designs roughly like early timesharing mainframes. For the purposes of security, our computers have two relevant features: memory management and privileged modes. Memory management was designed to keep processes from interfering with each other and the operating system. You really don't want someone else who is logged into another terminal (a device capable of displaying 24 lines of 80 characters each, a keyboard, and a serial interface maxing out at 19,200 baud [1]) writing into your process's memory, and especially not the operating system's memory. Note that terminals on early systems were often Teletype Model 33 ASRs, capable of uppercase only but also allowed input or output via a paper tape reader/puncher [2]. Teletypes used Baudot, just five bits per character, with a maximum rate of 10 characters per second. I actually used Teletypes on a Multics system in 1969.

Memory management on mainframes in the 1970s didn't always work well. In 1979, I crashed a mainframe, using a DECwriter 300, a much nicer and quieter terminal, while taking an operating systems course. I made a mistake entering a substitute command and only noticed when the command was taking forever to complete. I had created an infinite loop, essentially replacing each character with two copies of itself. What clued me in to what I had done was when other people in the terminal room started getting up and leaving, knowing it would take at least 20 minutes to reboot the mainframe.

In our "modern" systems, memory management works very well at protecting processes from one another, and events like the one I just described won't happen. This is where privilege mode [3] comes in. Because the operating system needs to be capable of doing things that normal processes cannot, the CPU switches into privilege mode when executing operating system code. While in privilege mode, the executing software can read or write anywhere in memory. Needless to say, this has made writing kernel exploits extremely popular. And as kernels are the largest single images with the most complex code (think multiple threads, locking, and device drivers) you generally run, there are lots of vulnerabilities to be found.

You might be wondering why we rely on such ancient mechanisms as the hardware basis for our security. Think of these mechanisms as being like internal combustion engines: they've been around a long time and have gotten fantastically more efficient. There are other reasons for using these mechanisms: they are familiar to both CPU designers and programmers (see People, below).

There was an alternative design, a mainframe built in the mid-to-late '60s: the GE 645 (later Honeywell). The GE 645 had segment registers, essentially hardware used to add an offset to each memory address. Unlike memory management, segment registers as used in this design weren't limited to large page sizes, so it was possible to have programs that could treat

different areas of memory as if they were different physical compartments, and limit access to those compartments. Please read my interview with Peter G. Neumann in this issue for more on segmentation.

Intel's flagship CPUs in the later '80s also used segment registers as a method for extending memory from 64K to 640K. Later incarnations of segment registers in Intel CPUs were more flexible and used in early hypervisors.

What's neat about being able to segment memory is that it becomes possible to protect regions *within the kernel* by making them read-only, and control access to other kernel regions. Currently, our operating systems are write-anywhere, leaving them ripe for exploitation. User-level programs can also use segments, so they can have code within a single process that operates with different sets of privileges. I suggest reading about CHERI [4], an example of a modern design that has segment registers as a key feature.

Software

At the time I am writing, Equifax has been hacked, and all of the data needed to steal 143 million US identities has been downloaded. Equifax has blamed the Apache Struts software for the exploit, even though they allegedly failed to install the update that would have prevented the attack. Obviously, even if Equifax's hardware had hardware features only available in the future, a Web script that allows customers making credit queries to access their database would still allow this attack. After all, requesting credit queries needs to work in this application.

Of course, attacks like this succeed because the software actually supports doing things, like downloading 143 million credit reports, through mistakes in design.

I've selected many papers about how to improve software design, the most recent appearing in the Fall 2017 issue [5]. Essentially, using carefully designed parsers as well as protocols that allow simple parsers (no looping or recursion allowed) would eliminate attacks like this. The parsers pass safe arguments to other routines for execution instead of any old thing that a clever attacker can slip through. There are companies that focus on reverse engineering input parsers, so they can report the exploitable weaknesses in the code they are processing (Veracode, for example), so making money honestly from coding mistakes in parsers is already a successful business [6].

People

Last but not least on the list of why our systems are insecure are people: the people who manage the systems and the people who program their software.

If I had been working as the CSO of a large financial company whose main business had to do with identity records, I would have insisted on having simple parsers, but also a gateway, a type of application firewall, that would limit access to the database to the expected queries, and rate limit the number of responses permitted.

Of course, I've left out an important aspect in my imaginary scenario: other people. Those other people might be programmers who don't understand what I have in mind, those topics having not been covered in any course. The programming of safe parsers is actually not something most people can do properly, which is why there are tools for doing this [5].

Other people who would balk at adding what would, in hindsight, turn out to be saving-the-business-important security would be managers and C-level executives, who would point out that adding security would "cost money" and "take time." Well, those people do need to balance risk against potential income, even though ignoring security has caused companies to go out of business.

The Lineup

We start off this issue with four articles about pretty basic stuff that people commonly get wrong. First up is Pearce et al., who examine the prevalence of DNS spoofing. They carefully designed a way to test whether DNS resolvers were lying, and found that a significant number of countries, most often but not always repressive regimes, did falsify results of DNS queries.

Next up, Chung et al. look at just how well people are doing at DNSSEC, the protocol for providing cryptographic proof that DNS queries return accurate results. The answer is: not well at all. Only a tiny fraction of domains use DNSSEC, and a very small fraction of that fraction have done it correctly. Part of the reason for this problem is design (DNSSEC is complicated, although just reading their article did more to help me understand DNSSEC than anything else I've read). There are other problems, like registrars that either do not accept the hashes ("DS" records) required to prove the correctness of their registered, second-level domains, or do so insecurely. To top that off, few resolvers actually check the correctness of the DNSSEC records they have downloaded.

Mayer and the team at SBA Research examine another mainstay of Internet security: HTTPS. They set up a user study where they asked more senior college students who allegedly could perform system administration to set up HTTPS securely. Hint: the vast majority of people are better off using https://letsencrypt.org/.

The last article in this series, where the ability of people to behave securely is in question, is about checking password strength. Melicher et al. developed a neural network small enough to download as part of a Web page, and proved their tool

(https://github.com/cupslab/password_meter) works much better than the current crop of tools, which reward people for capitalizing the first letter and ending their password with "!".

Bano, Al-Bassam, and Danezis volunteered an article about fixing the Bitcoin blockchain. The Bitcoin blockchain can only handle a few transactions per second and takes at least 10 minutes before those transactions can be committed. Talk about a failing database, even if it uses strong public key encryption for security. Bano and his co-authors explain several alternative methods for increasing the performance of blockchains.

I decided to interview Peter G. Neumann for this issue. He was part of the design team for Multics, worked on several design papers for better security, and is part of the CHERI team for improving hardware security. Peter is fun to listen to, a great storyteller, and someone who has been involved in some amazing work.

Gu and co-authors from the EECS Department at the University of Michigan provide the lone Systems article in this issue. Their modification-free solution for memory disaggregation, INFINISWAP, takes advantage of RDMA to share unused memory in a cluster of systems as faster swap devices. Using the combination of a daemon and a kernel module that presents a block device interface, INFINISWAP improves performance over swap on hard drives for paging while still providing reliability, a pretty cool idea.

In the SRE and Sysadmin section, John Looney writes about psychological safety for SRE teams. Using his experience as part of a specialized team that studied SRE teams, John makes great use of examples of how not to support team members, then shows how things could have been done better.

Vladimir Legeza wanted to write about the difference between system administration and SRE. Through the use of examples, Vladimir lays out what he considers key differences between how the two groups work and what sysadmins could learn from the SRE way of doing things.

Kurt Lidl shares his knowledge of Docker. I liked this article a lot for the level of useful detail provided, as well as for the comparisons to other mechanisms for isolation of groups of processes.

David Beazley has written his final column for *;login:*. Appropriately, he wrote about exiting gracefully (in Python, although the double meaning is obvious). We will miss David for his careful and thorough explanations of Python.

David Blank-Edelman writes about Perl-without-Perl. While this might sound strange, there are a lot of times when you want a tool or script that processes a Perl script, perhaps just for extracting some portion of the script, without involving execution of the script. David covers a module and other tools for doing this.

Chris (Mac) McEniry explains how to use Hashicorp's Vault, a Go library used for storing secrets, such as the passphrases used to decrypt private TLS keys. Mac also includes the use of dep, a tool for managing Go dependencies.

Dave Josephsen describes how he used tcpdump to monitor network traffic of images his company is running in Amazon's cloud. Anyone who needs an effective way of monitoring network traffic when the network is run and controlled by someone else needs to read Dave's column.

Dan Geer writes an essay about Data with a capital D. Dan ruminates about the importance of the "Big Data" we collect, and explains the two things we should consider whenever we decide to collect data.

Robert Ferrell has written his humor column this time about third-party loss of personal data and risk mismanagement.

We have three book reviews this time, one by *;login:*'s Managing Editor, Michele Nelson, and two by Mark Lamourine.

Conclusion

I know that I have written about the failure of people in the past. I also know that it's just not a good idea to expect most people to write software or manage systems securely when even experts do these tasks poorly. Computing is complex, abstract in many ways (how often have you looked at a heap?), and generally the people responsible for the software and hardware that becomes the most popular (think C) are geniuses or work with geniuses. Expecting the bulk of humanity to reach this level is simply unreasonable. Geniuses, by definition, represent a tiny fraction of the population.

Services like Let's Encrypt go a long way toward removing the requirement that everyone who wants to use HTTPS needs to be an expert or a genius. We need to extend this type of service to include programming languages, system management, and improved hardware-based security if we ever expect to have even moderately secure systems and a reliable Internet.

References

[1] Example of terminal: http://bit.ly/2fLWPdQ.

[2] Teletype 33 ASR: https://en.wikipedia.org/wiki/Teletype_Model_33.

[3] Microsoft on privilege mode: http://bit.ly/2yD6f3l.

[4] CHERI: http://www.cl.cam.ac.uk/research/security/ctsrd/cheri/.

[5] G. Couprie and P. Chifflier, "Safe Parsers in Rust: Changing the World Step by Step," *;login:*, vol. 42, no. 3 (Fall 2017): https://www.usenix.org/publications/login/fall2017/couprie.

[6] Veracode sold: http://bit.ly/2lX8TtT.

SAVE THE DATES!

SRECON® AMERICAS

MARCH 27–29, 2018 • SANTA CLARA, CA, USA
www.usenix.org/srecon18americas

SRECON® ASIA AUSTRALIA

JUNE 6–8, 2018 • SINGAPORE
The Call for Participation will be available in December 2017.
www.usenix.org/srecon18asia

SRECON® EUROPE MIDDLE EAST AFRICA

AUGUST 29–31, 2018 • DUSSELDORF, GERMANY
The Call for Participation will be available in February 2018.
www.usenix.org/srecon18europe

SREcon is a gathering of engineers who care deeply about site reliability, systems engineering, and working with complex distributed systems at scale. It strives to challenge both those new to the profession as well as those who have been involved in it for decades. The conference has a culture of critical thought, deep technical insights, continuous improvement, and innovation.

Follow us at @srecon

usenix®
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION