;login:

SUMMER 2018 VOL. 43, NO. 2

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

2018 USENIX Annual Technical Conference

July 11–13, 2018, Boston, MA, USA
www.usenix.org/atc18

Co-located with USENIX ATC '18

HotStorage '18: 10th USENIX Workshop on Hot Topics in Storage and File Systems
July 9–10, 2018
www.usenix.org/hotstorage18

HotCloud '18: 10th USENIX Workshop on Hot Topics in Cloud Computing
July 9, 2018
www.usenix.org/hotcloud18

HotEdge '18: USENIX Workshop on Hot Topics in Edge Computing
July 10, 2018
www.usenix.org/hotedge18

27th USENIX Security Symposium

August 15–17, 2018, Baltimore, MD, USA
www.usenix.org/sec18

Co-located with USENIX Security '18

SOUPS 2018: Fourteenth Symposium on Usable Privacy and Security
August 12–14, 2018
www.usenix.org/soups2018

WOOT '18: 12th USENIX Workshop on Offensive Technologies
August 13–14, 2018
www.usenix.org/woot18

ASE '18: 2018 USENIX Workshop on Advances in Security Education
August 13, 2018
www.usenix.org/ase18

CSET '18: 11th USENIX Workshop on Cyber Security Experimentation and Test
August 13, 2018
www.usenix.org/cset18

FOCI '18: 8th USENIX Workshop on Free and Open Communications on the Internet
August 14, 2018
www.usenix.org/foci18

HotSec '18: 2018 USENIX Summit on Hot Topics in Security
August 14, 2018
Lightning Talk submissions due June 11, 2018
www.usenix.org/hotsec18

SREcon18 Europe/Middle East/Africa

August 29–31, 2018, Dusseldorf, Germany
www.usenix.org/srecon18europe

OSDI '18: 13th USENIX Symposium on Operating Systems Design and Implementation

October 8–10, 2018, Carlsbad, CA, USA
www.usenix.org/osdi18

LISA18

October 29–31, 2018, Nashville, TN, USA
www.usenix.org/lisa18

Enigma 2019

January 28–30, 2019, Burlingame, CA, USA
Submissions due August 22, 2018
www.usenix.org/enigma2019

FAST '19: 17th USENIX Conference on File and Storage Technologies

February 25–28, 2019, Boston, MA, USA
Submissions due September 26, 2018
www.usenix.org/fast19

NSDI '19: 16th USENIX Symposium on Networked Systems Design and Implementation

February 26–28, 2019, Boston, MA, USA
Paper titles and abstracts due (Fall deadline) September 13, 2018
www.usenix.org/nsdi19

SREcon19 Americas

March 25–27, 2019, Brooklyn, NY, USA

USENIX Open Access Policy

USENIX is the first computing association to offer free and open access to all of our conferences proceedings and videos. We stand by our mission to foster excellence and innovation while supporting research with a practical bias. Your membership fees play a major role in making this endeavor successful.

Please help us support open access. Renew your USENIX membership and ask your colleagues to join or renew today!

www.usenix.org/membership

;login:

SUMMER 2018 VOL. 43, NO. 2

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Michele Nelson
michele@usenix.org

COPY EDITORS
Steve Gilmartin
Amber Ankerholz

PRODUCTION
Arnold Gatilao
Jasmine Murcia

TYPESETTER
Star Type
startype@comcast.net

USENIX ASSOCIATION
2560 Ninth Street, Suite 215
Berkeley, California 94710
Phone: (510) 528-8649
FAX: (510) 548-5738

www.usenix.org

;login: is the official magazine of the USENIX Association. *;login:* (ISSN 1044-6397) is published quarterly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for nonmembers are $90 per year. Periodicals postage paid at Berkeley, CA, and additional mailing offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

EDITORIAL

Musings

RIK FARROW

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

While I was at OSDI in Atlanta (2016), I heard a couple of paper presenters mention the use of machine learning (ML) as part of their research for those papers. I was immediately intrigued because ML, part of artificial intelligence (AI), seemed much too complicated to simply drop in as part of a graduate research project. So I set out in search of someone who could write an ML article for beginners.

I first asked Mihai Surdeanu [1], whom a fellow attendee at OSDI suggested I contact. Surdeanu considered the possibility of writing an article but backed out because he was already too busy. He suggested a list of names in the machine learning field. I tried Andrew Ng, the person who developed the famous Coursera class on machine learning, and of course he was too busy. Then I tried his course, but soon found myself flummoxed by the math. I had taken three years of advanced calculus in college, but that was 40 years ago. And I hadn't taken the class on linear regression, which turned out to be as important as matrix manipulation early in Ng's course.

Mihai comforted me, saying that Ng's class is for people designing new ML algorithms and that most people just use ready-made ones. That got me thinking: perhaps I could read some books and learn enough about ML to be dangerous.

The first book I found was *Machine Learning in Python* by Michael Bowles [2]. I also found a PDF of the book online, along with the code examples and data. That was good because I was in a hurry.

Bowles' book focuses on just two classes of ML algorithms: penalized linear regression (e.g., elastic net and lasso) and ensemble methods (e.g., stacking and gradient boosting). Bowles has an easy-to-follow writing style, and it's in Chapter 2 of his book where I learned about the importance of understanding the data you want to use for training data for your ML model. If you use bad data, your model will turn out to be useless. Your data also determines which class of algorithm is most appropriate. Bowles spends a lot of time performing statistical analyses of his data sets, helping me understand the importance of this step.

I began to understand why no one—and yes, I had asked more than just the two people mentioned here—wanted to attempt to write a "beginners guide" to ML. There really is a lot involved.

Mihai suggested a newer book, *Deep Learning with Python* by Francois Chollet [3], saying that Chollet uses newer algorithms than the Bowles book, which is two years older. I think I will continue with Bowles for now, then move on to Chollet.

I came out of my ML adventure realizing just how important training ML data is. Garbage-in, garbage-out was an old programmer saying, and it's just as relevant today as it was when I learned it decades ago. The goal of ML is to produce a model that can be used to predict output given new data. The ML algorithms build these models through a recursive process, refining, for example, weights for features that guide the decision process.

Stayed tuned, as I don't plan on giving up yet. I've heard there will be an extreme need for programmers who understand ML in the near future...

The Lineup

When I heard about the Cyber Grand Challenge (CGC) during USENIX Security '16 in Austin, people there suggested watching a long video. I was too impatient to ever get very far into the video. Other articles about CGC were rehashes of press releases: all alike without much info. But David Brumley, CEO of ForAllSecure and, through CMU, mentor to the hacking team PPP, had a participant's first hand perspective. His team not only won the CGC, it was the first autonomous computer system ever to compete in a DefCon Capture the Flag, making him the ideal person to describe the contest; ForAllSecure's Mayhem program was up against the best human hacking teams at DefCon and did very well.

Travis McPeak had also given a talk at Enigma, but I wanted more details and so engaged Travis in an interview about how they handle least privilege at Netflix. Netflix uses AWS, which in turn provides extensive configuration for controlling permissions. Travis makes points about mistakes with granting permissions, problems with setting up permissions when an app is first deployed, and how to automatically remove permissions from apps over time.

I also interviewed Swami Sundararaman, one of the co-chairs of the new HotEdge workshop (associated with USENIX ATC '18). I wanted to know *why*, suddenly, the edge is hot, and *what* this edge consists of. Turns out that, of course, things change, and there are new demands best met by services that will live at the *edge* of the Internet.

I was strolling through the FAST '18 posters and came across some familiar faces talking about something that just seemed too weird. Philip Kufeldt and his associates from many different storage companies were introducing *eusocial* devices. Like ants or bees, one device alone can't do much. But together, they can accomplish a lot. In this case, they want to take over data management to help unload CPUs and the memory pathways from work that could be handled by the eusocial storage devices.

Haryadi Gunawi et al. had presented a very cool paper at FAST about research into slow failures. When memory, or a networking or storage device, fails quickly, the failure is obvious. But when the failure is partial, such as a slowdown but not death, uncovering the failed component is much harder, particularly in current layered architectures.

Sean Kamath at LISA17 said he wanted to write another article about LISA. His first, "Whither LISA" [4], written in 2010, disturbed marketing folks but did stir up discussion. The LISA conference is undergoing more changes this year [5], and Sean has written a retrospective of his 25 years of attending LISA, how things have changed, and why they will be different in the future.

Mac McEniry continues to evolve the remote execution example with the addition of command-line processing and execution of multiple modules on the server side. Mac introduces Cobra, a Go package that is more flexible than the native Golang interface to the command line.

David Blank-Edelman takes a look at using graph databases from Perl. He focuses on Neo4j, explaining how graph databases work through examples, then covers who to do this via the REST::Neo4p module.

Dave Josephsen, as excited as ever, interviews Matt Broberg, the VP of Community for Sensu Inc. Sensu is an open source monitoring system that by design is very flexible and is already popular. After introducing Sensu 1.x, Dave moves on to asking about Sensu 2.0, written in Go, and how that will change things for Sensu users.

Dan Geer and Michael Roytman write about recall and precision, terms defined in their article. Their interest lies in how a security practitioner finds the crucial nuggets among all the events being generated by their security monitoring applications. They show, through a hypothetical example, how an organization couldn't possibly uncover all the significant events, and suggest how security software should be improved. Apart from the work reflected in this column, Michael turns out to have a very amazing project [6].

When I mentioned my interest in machine learning, Robert Ferrell wanted to participate too. Robert focuses on some of the failures of the not-so-smart machines around his house.

Mark Lamourine has reviewed three books this time. The first is *Teach Yourself Go in 24 Hours*, a Sams book that Mark was favorably impressed by. The second seemed a bit of a stretch for our community, but Mark came away from reading *Crucial Conversations* feeling there was a lot to learn there. Mark also read a relatively thin tome, *Linux Hardening*, and while he yearned for a bit more depth, concurs that the author succeeds by following a narrow path.

While I've been interested in AI and ML for several years now, a propaganda video about the dangers of autonomous killer drones (little tiny ones) really motivated me. We can expect to see more autonomous weapons going forward: just search for *autonomous paintball sentry gun*. People have been building these for years, and the South Koreans have taken their version a bit more seriously, replacing the paintball guns with machine guns.

But these are a far cry from little drones that can recognize their targets, with each drone working cooperatively, so that two drones don't gang up on one target, then approaching and killing the victim. The video rather loses the point that you can swat a little drone from the air with your hand, and certainly would do so before it got close enough. When current image recognition

software still has problems describing what's in a photograph, forget about drones recognizing specific faces, much less cooperating during an attack.

The more common concern these days about AI is what is termed *general intelligence*. General intelligence is what humans are supposed to be able to manage: flexible behavior in varying circumstances, the ability to communicate intelligently, and so on. If you note a bit of cynicism here, just consider world politicians today. General intelligence in AI is supposed to lead to the extinction of humanity.

General intelligence in AI is as far off today as fusion power. Instead, AI works best when trained to operate in a particular, narrow field: for example, playing Go or providing medical advice; the game-playing Watson has moved on [7]. And to be honest, I'd much rather believe in a far-off future like the one created by Iain M. Banks in his Culture series. While Banks' AI machines (called *minds*) have godlike intelligence, they choose to continue working with humans—for reasons we might consider inscrutable.

Today, our AI still struggles with speech comprehension, and our autonomous paintball sentries don't even need AI. Instead, we need to focus on what ML can do to help us to understand the mountains of data we are acquiring every day.

References

[1] Mihai Surdeanu: http://www.surdeanu.info/mihai/.

[2] M. Bowles, *Machine Learning in Python* (Wiley, 2015), ISBN: 978-1-118-96174-2.

[3] F. Chollet, *Deep Learning with Python*: (Manning, 2018), ISBN-13: 978-1617294433.

[4] S. Kamath, "Whither LISA," *;login:*, vol. 35, no. 1 (February 2010): https://www.usenix.org/system/files/login/articles/102-kamath.pdf.

[5] LISA18: https://www.usenix.org/conference/lisa18.

[6] A. Maxmen, "Out of the Syrian Crisis, a Data Revolution Takes Shape," *Nature*, October 25, 2017: https://www.nature.com/news/out-of-the-syrian-crisis-a-data-revolution-takes-shape-1.22886.

[7] "IBM's Watson AI Recommends Same Treatment as Doctors": https://futurism.com/ibms-watson-ai-recommends-same-treatment-as-doctors-in-99-of-cancer-cases/.

Letter to the Editor

Rik,

In your Spring 2018 *;login:* column you asked about references to disk (controllers) taking over block/sector placement. The first I heard of this was in the early 1990s; after a bit of hunting I located a paper that would correspond to what I remember:

Robert M. English and Alexander A. Stepanov, "Loge: A Self-Organizing Disk Controller," in *Proceedings of the USENIX Winter 1992 Technical Conference,* pp. 237–251.

I don't remember who I heard present this when I was working for DEC Networks AD.

I found the paper online at http://stepanovpapers.com/Loge.USENIX.pdf (it appears to be just a little too old to be in USENIX's online archive, and I haven't found an entry for it in ACM's portal).

Cary Gray

Rik responds:

That's an amazing find. And I'm guessing that lots of people don't know that Hewlett-Packard manufactured disk drives in the past.

Although there are no links from the USENIX site yet, old proceedings are being digitized and hosted on archive.org. You can find the full Proceedings of the Winter 1992 Annual Technical Conference here: https://archive.org/details/winter92_usenix_technical_conf.

Thanks!
Rik

Register Today!

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

2018 USENIX Annual Technical Conference

JULY 11–13, 2018 • BOSTON, MA
www.usenix.org/atc18

The 2018 USENIX Annual Technical Conference will bring together leading systems researchers for cutting-edge systems research and the opportunity to gain insight into a wealth of must-know topics, including virtualization, system and network management and troubleshooting, cloud and edge computing, security, privacy, and trust, mobile and wireless, and more.

The program includes a Keynote Address by Dahlia Malkhi, *VMware Research*, 76 refereed paper presentations, a poster session, Birds-of-a-Feather sessions (BoFs), and more.

Co-located with USENIX ATC '18

HotStorage '18

10th USENIX Workshop on Hot Topics in Storage and File Systems
July 9–10, 2018
www.usenix.org/hotstorage18

Researchers and industry practitioners will come together for this two-day workshop on the cutting edge in storage technology and research and explore and debate longer-term challenges and opportunities in the field.

HotCloud '18

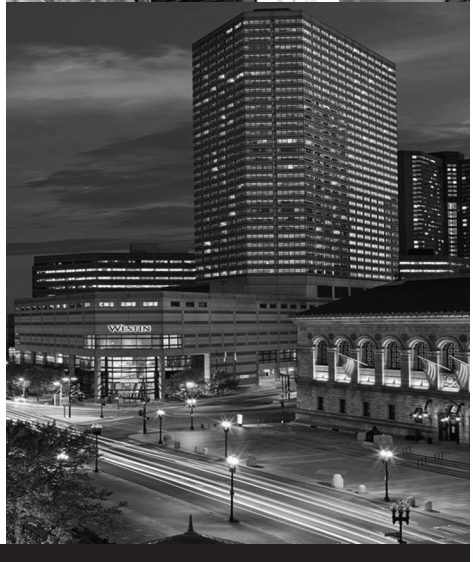
10th USENIX Workshop on Hot Topics in Cloud Computing
July 9, 2018
www.usenix.org/hotcloud18

HotCloud brings together researchers and practitioners from academia and industry working on cloud computing technologies to share their perspectives, report on recent developments, discuss research in progress, and identify new and emerging trends in this important area. While cloud computing has gained traction over the past few years, many challenges remain in its design, implementation, and deployment.

HotEdge '18

USENIX Workshop on Hot Topics in Edge Computing
July 10, 2018
www.usenix.org/hotedge18

Join researchers and practitioners from academia and industry to discuss work in progress, identify novel trends, and share approaches to the many challenges in design, implementation, and deployment of different aspects of edge computing.



Register by June 19 and save!

The Cyber Grand Challenge and the Future of Cyber-Autonomy

DAVID BRUMLEY

David Brumley is the CEO and co-founder of ForAllSecure, and a Professor at Carnegie Mellon University in ECE and CS. ForAllSecure's mission is to make the world's software safe, and they develop automated techniques to find and repair exploitable bugs to make this happen. Brumley's honors include a United States Presidential Early Career Award for Scientists and Engineers (PECASE), a Sloan Foundation award, numerous best paper awards, and advising one of the world's most elite competitive hacking teams.
dbrumley@forallsecure.com

The Cyber Grand Challenge was about much more than a capture-the-flag (CTF) competition between computers. The people who built those systems had to learn how to replicate the behavior of human hackers, perform binary program analysis, patch vulnerable applications—or not, if installing the patch hurt performance or resulted in a functional regression. In this article, based on my 2018 Enigma talk [1], I will describe the competition, and also how human hackers and the CGC competitors' systems have different strengths.

I want to begin by introducing you to the person whom I believe is one of the world's best hackers. His name is Loki and he's an expert at web browser security. At the 2016 Pwn2Own competition [2], Loki demonstrated three new vulnerabilities and was able to exploit them in applications that would have enabled him to break into 85% of the computers in the world.

The rules for Pwn2Own are actually pretty simple. The people running the contest install a fully patched version of an operating system on a laptop, and then they install the latest, greatest, direct-from-developers version of a web browser. The goal is to break into the computer through the web browser. If you think about it, this is pretty amazing because vendors spent a lot of money trying to secure their operating systems. But Loki has been studying computer security, is an expert in the internals of Google Chrome, and has been studying web browsers for a long time.

Loki sat down in front of a laptop running Google Chrome on top of Windows and within two minutes had demonstrated a vulnerability. What's amazing is that over the course of the next two days he also demonstrated new zero days in Microsoft Edge and Apple Safari. Those were the three vulnerabilties that would have allowed him to break into 85% of the world's computers.

Loki is the world's best hacker, in my opinion, but he's not a criminal. I don't want to conflate the terms hacker and criminal. Loki responsibly disclosed those vulnerabilities to vendors, and those vendors issued updates that protected millions of people.

Over the course of that weekend Loki also made $145,000. Not bad for 15 minutes of work, keeping in mind that like a professional athlete, Loki spent a lot of time preparing for this contest.

Software and Vulnerabilities

Think of all the software that you use every day. And I'm not just talking about the software that's running on your computer, your laptop, or on your smartphone. I'm also thinking about all this software that you interact with on nonobvious devices: IoT devices, your WiFi router, even the software that powers the safety system on your car. Who's checking it for security vulnerabilities?

How do we go about doing what Loki does and do it at scale? On just the Google Play and Apple stores, a new app is released every 13 seconds. How do we go about checking software when a new app is released so frequently? I've been working on this problem for a long time, along with other academic researchers. If we could take what Loki does and program computers to emulate it, computers could do that work for us.

At Carnegie Mellon University, we built a tool called Mayhem that takes off-the-shelf software and audits it for vulnerabilities. As an example, we used Mayhem to explore `iwconfig`. Mayhem systematically explored the state space in `iwconfig` and output an exploit. You can take that exploit, give it as input to `iwconfig`, and get a root shell. At a basic level, we enabled a computer to take a software binary, autonomously find a security vulnerability, and prove it with a working exploit. This isn't the world that we live in today, but I think this is the world that we need to live in—one where not just developers can check for the security of applications but anyone can, using systems like Mayhem.

Scale

In one study we used Mayhem to examine 37,391 programs—every Debian program available. We spent three years of CPU time analyzing the programs: that amounts to five minutes per application. We did this in an embarrassingly parallel way by bringing up a bunch of Amazon nodes. We found 2.6 million new crashes due to 13,875 new bugs in those programs. Of those, 250 exploits would allow getting a shell.

We want to be operating at the scale where we can check the world's software for exploitable bugs. Doing analysis like this cost us 28 cents per new bug and $21 per exploit. Compare that to Loki who made $145,000 for three working exploits.

Cyber Grand Challenge

We've been doing this research at CMU for over a decade, along with other researchers such as Giovanni Vigna at UC Santa Barbara, Dawn Song at UC Berkeley, and a much larger community. DARPA, the Defense Advanced Research Projects Agency, stepped up to challenge this community in an open forum. DARPA issued the Cyber Grand Challenge, wanting to do for cyber what the autonomous driving challenge [3] did for vehicles. DARPA wanted to turn cyber into something that was completely autonomous and controlled by computers. They challenged the community to combine the speed and scale of automation with reasoning abilities that exceeded those of human experts.

DARPA first issued an open call for proposals for a fully autonomous offense and defense contest, with a $2 million cash prize for the winner. That got lots of attention, and over 100 US entities registered for the CGC. At the end of the first year, DARPA pared down the entrants based on the same performance factors to be used in the final contest, leaving seven contestants.

The final contest occurred at DefCon 2016. This contest was different from the usual Capture the Flag (CTF), held at DefCon every year, which pits human teams against one another. In the CGC, it was computer against computer, with an air gap separating the computers from any attempt at outside help or interference.

DARPA structured the event by sending programs to all the contestants' systems. These systems needed to find vulnerabilities in those programs and create patches that fixed those vulnerabilities, sending patches back to the DARPA moderator. The DARPA moderator system evaluated the security solutions based on a number of factors. The first was if you created an exploit, does it work against other people's patched binaries. DARPA called this "consensus evaluation": you get points based on whether your exploit works against other people's running programs—patched or not.

Second, the patch itself is evaluated for security, checking to see whether the patch itself could be exploited.

But the world isn't just about security. It's also about things like functionality. DARPA would take the patched binaries and make sure that the system retained all its original functionality. After all, if a patch breaks your system you're not going to install it. They also measured the performance of the patch. DARPA's moderator would look at things like memory overhead.

And so when you considered the type of contest, it wasn't all about security. It was about making decisions that operated within a confined space to make sure that it wasn't just the most secure but also maintained performance and functionality. To give you an idea, on our system, if we determined that our patches had more than 5% overhead, it may have been better to play the original buggy binary.

Round after Round

When a competitor found an exploitable bug and submitted a patch, things didn't end there, because that's not how the world works. DARPA changed the direction of the community by saying the goal here is to win, and the way you win is by giving people access to your patches. DARPA's moderator would take our patches and give them to our competitors, and the competitors could do further analysis to see whether they could circumvent them. They could try to steal our patches and use those patch techniques themselves and submit them back to the DARPA moderator round after round. So the CGC wasn't just about security and a point in time, but about security as it evolves. It allowed attackers and defenders to learn from each other. And by the end of the contest, we'd completed over 95 rounds to determine a winner.

When we looked at the scope of CGC, we had to do three things. First, we needed to be able to automatically exploit software. We had to do what Loki does to find vulnerabilities, and we had to teach a computer to do it. Second, we had to be able to automatically rewrite binaries to add in defenses to prevent them from being exploited again. And third, just as importantly, we had to make better decisions than our opponents. That was huge.

Automatic Vulnerability Discovery

Let's talk about how we went about doing the automatic vulnerability discovery. We needed to be able to perform code analysis without source code. We built the binary analysis platform (BAP, [4]) at CMU, available for free from GitHub. BAP allows us to take a binary and raise it up to an intermediate representation that is useful for doing program analysis.

We then created tools to find vulnerabilities in software using a technique called "symbolic execution." We, and others in the community, considered symbolic execution as a very academically promising technique that we could publish papers about while advancing the frontiers of research. But we discovered during this contest that trying to find the best single solution was the wrong thing to do.

Instead, we realized that applying a portfolio of techniques, such as fuzzing and crash exploration, would be more effective. If you have a crash, the program has some sort of mental problem with the world at that point. The program *thinks* the world is different than it actually is; if there's a bug in one place, there's likely a bug nearby as well. We also built a feedback loop between these techniques which allowed us to find far more vulnerabilities than any single technique alone.

At the end of the contest we found 67% of our bugs with fuzzing and 33% with symbolic execution. But those percentages only reflect the final uncovering of the bugs themselves. We found that the symbolic executer would often reach a promising part of a code then hand that over to a fuzzer, which used a different set of techniques. It would be the fuzzer that ultimately found the vulnerability, but only after being enabled by the symbolic execution technique. We learned that building a portfolio of techniques that work together cooperatively is always going to outperform any single technique.

Defense

For defense, we had to be able to statically rewrite binaries. We used data flow analysis as a basis to focus formal program analysis on understanding how a program worked. We could then derive an analysis that would rewrite the program. Here too we used the portfolio, with two techniques. The first we called hardening, rewriting the binary to essentially introduce seatbelts: control flow integrity, stack canaries, ASLR, DEP, and so on. Now these remediations are agnostic about whether there is a vulnerability or not, but they make the program safer overall. Second, when we found a particular bug, we'd automatically rewrite that portion of the code where the bug occurred to introduce safety checks.

After hardening the binary, we could add crash-specific patches for vulnerabilities when we found them. For example, one of the challenges DARPA gave us in CGC was based on the SQL Slammer worm. When Mayhem receives a binary, it immediately starts generating automatic regression tests, which function as the baseline for the binary. We'd start patching, creating a hardened binary and then replaying those test cases to make sure that we had no loss of performance and functionality. We would also go in and rewrite the patches, replaying those automatically generated test cases to measure and ensure that functionality and performance were maintained. Then we would have to decide which of these patches to apply. Instead of having just one patch, we had an array of patches based on the portfolio of techniques. We would measure them and empirically determine the best ones to deploy at a particular time. Finally, we had to build a system that was completely autonomous.

Dynamic Scaling

When DARPA started the contest, we didn't know whether they were going to give us 10 programs, 100 programs, or 1000 programs. So we had to build in the capability to dynamically scale our environment to dedicate resources where they were going to matter most. For example, if we have a program that we keep finding new bugs in, it would make sense to devote more resources to that than to a program that's not buggy.

Second, we had to make sure that we were playing the game the optimal way. For example, if we created a patch and that patch had 5% overhead, we might decide that it was too dangerous to play since that 5% overhead would hurt our score unless we thought someone was attacking us.

In a nutshell, we would do the binary analysis, we'd harden, we'd find exploits, we'd patch, and we'd run through a decision process that determined the best security solution. Then we'd deploy and iterate through this process again and again and again.

In the CGC, we faced some of the most notable names in program analysis out there: UC Berkeley and UC Santa Barbara, who have been doing research on this forever; Raytheon, a large defense contractor, was also in the final seven contestants. We also had a two-person team from the University of Idaho who qualified, beating out 93 other teams.

At the end of this contest, Mayhem won. ForAllSecure, the company we founded to continue the development, got the $2 million cash prize. But when you looked at the contest, everyone had little twists on their techniques that were different, and if you look at the scores you'll notice they're not very far apart.

CTF

At the end of the CGC, our system, Mayhem, got to participate in the annual DefCon 24 CTF [5] (Table 1 shows the final results). Just to give you an idea of the caliber of the people Mayhem played against, the number three team DEFKOR had Loki on it, the person who demonstrated three zero-days over the course of two days.

Team	Score
PPP	113555
b1o0p	98891
DEFKOR	97468
HITCON	93539
KaisHack GoN	91331
LC⚡BC	84412
Eat Sleep Pwn Repeat	80859
Binja	80812
Pasten	78518
Shellphish	78044
9447	77722
Dragon Sector	75320
!SpamAndHex	73993
侍	73368
Mayhem	72047

Table 1: DefCon 24 CTF results [5]

We had built Mayhem as part of ForAllSecure, and I've also been the faculty mentor of the human hacking team at CMU, PPP (Plaid Parliament of Pwning). So I have some insight into what the machines could do versus what the humans could do. Although the machine lost, you'll notice from the scores that it was competitive. For two out of the three days of the DefCon 24 CTF, the machine was beating some of the teams.

But there are differences. For example, if you look at the humans, they have an incredible notion of being able to abstract details (Table 2). It's something that humans are great at, while the machine has precision going for it. At one point in the contest, for example, aPPP was looking at a line of code they thought might be vulnerable but couldn't figure out how to exploit. Mayhem was able to create an exploit because it had to reason about program branches and the particular program state, which was extremely complicated and would far exceed human understanding. Mayhem was able to do that in a fraction of a second.

Second, humans have intuition, and this is extremely important when you hack because you have to decide where to focus your attention. You may think, this part of the code looks extremely tricky, and therefore I'm going to focus my attention there.

Machines have brute force. Very simply, brute force is incredibly useful when you're trying to analyze, exploit, and patch applications at scale. And the way we see it from our research point of view is that once a person has an intuition of where to look, brute force can be used as a leverage point.

Human	Machine
Abstraction	Precision
Intuition	Brute force
Creativity	Scalability

Table 2: Human vs. machine qualities when it comes to hacking, as well as other forms of problem solving

Finally, humans have creativity. The machine will only look for vulnerabilities that it has been programmed to look for, while humans aren't restricted. Attackers and defenders get to co-evolve. For example, there is a class of attacks called timing attacks. The way I describe them is as follows: suppose my wife asked me, "Do I look fat in these pants?" and I took a few seconds to respond. It doesn't really matter what my response is now. The amount of time it took me to respond reveals all the information needed. It's the same way in security, where the amount of time it takes to do something can reveal something about the secret.

Humans have this great creativity to invent new classes of attacks. While the machine, once we program it to look for that class of attacks, has enormous scalability in looking at all the programs in the world.

Conclusion

In this article, there have really been two themes. The first theme is that human effort alone does not scale. Apps are being released at a pace that far outstrips people's ability to examine every one. Yet we need something as a safety checkpoint to make sure we've looked at every piece of software for security vulnerabilities. It's just too important not to do.

Second, we can teach computers to hack. Humans can teach computers to do at least a little bit of what Loki does and apply that to every piece of software in the world.

References

[1] Enigma 2018 talk: https://www.usenix.org/conference/enigma2018/presentation/brumley.

[2] A. Armstrong, Pwn2Own 2016—The Results: http://www.i-programmer.info/news/149-security/9556-pwn2own2016.html.

[3] https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles.

[4] Binary Analysis Platform: https://github.com/BinaryAnalysisPlatform/bap/graphs.

[5] DefCon 24 CTF: https://www.defcon.org/html/defcon-24/dc-24-ctf.html.

Interview with Travis McPeak

RIK FARROW

Travis works at Netflix on the Cloud Security team. He enjoys building automation to increase security while boosting developer productivity. Travis is a core developer of the Bandit and Repokid open source projects and has presented at security conferences, including BlackHat USA 2017, Enigma 2018, and re:Invent 2017. He currently serves as the Bay Area Open Web Application Security Project chapter leader. In previous roles he has served on the OpenStack Security team and as a founding member of the Cloud Foundry Security Team.
travis.mcpeak@gmail.com

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

I missed attending Enigma this year, so I started watching the online videos as soon as they became available. As always, I was doing more than just satisfying my own curiosity—I was also looking for talks that deserve a wider audience. I found several, asked the speakers, and Travis McPeak was the first to respond.

The technique of least privilege goes back to the dawn of computer security. First published in 1973, and presented at the fourth SOSP, Saltzer and Schroeder [1] laid out the ideas for granting only the level of privilege needed so that a particular application can function as intended.

Travis McPeak adds a new spin on this technique by applying it to applications deployed in the AWS cloud, starting with a safe list of permissions and automatically removing unused permissions over time, using an application he shares on GitHub.

Rik Farrow: What is least privilege, and why is it so hard for developers to get right?

Travis McPeak: Least privilege is a classic case of a simple concept that is very difficult to apply at scale. The idea itself is intuitive: we should only give applications the permissions that are required to function correctly. This is useful because in the case that an application becomes compromised, we can constrain the potential impact. For example, we protect files on Linux systems by granting read/write/execute permissions to only the user or group that needs access. This protects application resources from other processes on the system that we may not trust. Browsers enforce tab-level isolation so that a compromised tab can't affect the confidentiality or integrity of other tabs. One important point about least privilege is that it's a moving target. As an application changes, permissions may need to be added or removed to match the new requirements. This is analogous to applications that require and then shed root privileges when no longer needed.

The problems with manually applying least privilege become increasingly apparent when an organization grows in size and complexity. If I work by myself on an application, it's pretty easy for me to add permissions as I need them. Removing permissions when they are no longer required is a bit more challenging as there is no built-in reminder or incentive to remove them. I can set periodic reminders for myself and remove things that aren't needed anymore, but this may not be my top priority. However, what about organizations with dozens of applications and hundreds of developers continuously developing and redeploying? Who is responsible for periodically cleaning up permissions?

In many organizations the security team is responsible for granting and revoking privileges. There are a few problems with this. The security team doesn't work on all of these applications, so they don't know when an application changed and no longer needs some of the permissions. Manual security reviews are also a big problem because security teams are trying to balance lots of high priority work with a relatively small number of staff. If security teams are expected to stay on top of application changes and manually adjust permission sets to least privilege, they may not have enough time left to perform other critical work such as

applying patches, building tools to support secure development, and reviewing applications for security vulnerabilities.

For these reasons security teams seem forced into a binary decision: if the application is "important" enough, manually review it, otherwise ignore it. Every application that is "important" and manually reviewed saps time from both the application's developers and the security team. Every application that's ignored presents a risk to the business.

RF: In your Enigma '18 talk [2], you mention an AWS mechanism for controlling privilege. Could you describe that mechanism?

TM: At Netflix we rely on Amazon Web Services (AWS) heavily as our cloud provider. AWS provides a powerful access-control system called Identity and Access Management (IAM) that gives us very fine-grained control over specific actions and the resources they apply to. Here's a simple example policy:

```
{
 "Effect": "Allow",
 "Action": ["s3:GetObject", "s3:PutObject"],
 "Resource": "arn:aws:s3:::example_bucket/example_path"
}
```

This policy statement grants read and write access to the "example_path" in the S3 "example_bucket." While this policy is simple, it can quickly become difficult to determine which actions and resources should be allowed in a policy. With thousands of permissions, AWS can be configured very granularly compared to Linux file permissions. This configurability makes it both a powerful tool for security teams and very complicated for regular users. Even IAM experts may have difficulty determining exactly which permissions are needed to support a given application workflow.

RF: What techniques did the security team at Netflix come up with to deal with maintaining or improving least privilege in their applications?

TM: We use data about the permissions and resources that are actually used by an application to remove permissions that aren't required. To understand how this works in our environment it is useful to track the life cycle of an application and its permissions, beginning with how it gets permissions in the first place. Rather than wasting the valuable time of both developers and the security team, we automatically grant, by default, most of the benign permissions that applications need to perform common tasks. When a developer creates a new application, an application-specific role with the default permissions is created on their behalf by our deployment tool. If the application needs to perform any unusual or potentially dangerous actions, the security team and developers will perform a manual review, but in most cases the default permissions allow the application to do everything it needs.

After the application has been launched, we begin profiling it with tools that collect the data that AWS provides about which permissions and resources are actually used. Once a threshold of time has passed, unused permissions are automatically removed. Our open-source tool Repokid [3] automatically calculates new policies that preserve used permissions, removes unused permissions, and rewrites the new policy over the old. This approach eventually generates perfect least-privilege policies because anything that is kept is, by definition, actually used by the application.

If developers require a new permission or need something that was previously taken away added back, the security team manually adds it, but this is done with a quick conversation rather than the manual security reviews that we used to have to perform. The reasoning is simple: if developers are asking for a permission, then they either need it and will use it, or it will be automatically removed.

Once an application stops being used entirely, Repokid will remove all of the permissions, and the role becomes powerless. This is important because unused and unmaintained applications are huge headaches for security teams. The old applications have the same permissions with which they were originally deployed but aren't receiving patches or attention. By automatically removing permissions from these unmaintained applications, we can close a huge security hole.

RF: What are the challenges to this approach and how do you address them in your solution?

TM: Some applications don't regularly use their permissions but need them on an infrequent basis. Our usage-based analysis fails for these applications, and if we aren't careful, we can break them. For this reason, it's important to identify such applications and exclude them. Fortunately, there are relatively few, and we can fall back to the traditional manual review process for them.

Another concern is the eventuality that we will break something. We have put a lot of thought and consideration into how we can recover quickly and also detect as rapidly as possible that we have broken an application. Our goal is to use high quality data sources to avoid breaking applications. If we do break something we detect it, and when we detect a broken application we can fix it with the push of a button by rolling back to an earlier permissions state.

At Netflix we are focusing on logistics such as how to inform developers when changes are occurring for their applications and to give them options to defer or entirely block changes. Our goal is for developers to view this as a service that the security team provides for them to automatically make their applications more secure. The better we can communicate this message, the more successful our program will ultimately be.

RF: Does this approach only work for AWS or can it be applied to other areas of security?

TM: The solution we developed is for AWS applications and permissions, but we think this approach extends equally well to other areas of security. For example, it should be possible to use the same kind of data to constrain application container permissions to allow only the required syscalls and capabilities. Applications running directly on Linux systems may similarly be constrained by AppArmor or seccomp profiles that are generated automatically based on usage profiling. Mobile application privileges may be reduced by profiling usage in a sandbox environment and then having required permissions suggested, taking the guesswork out of permissions for the application's developer. There are many other possibilities for a similar approach applied to other areas of security in the future.

RF: What are the next steps for the project and its application at Netflix?

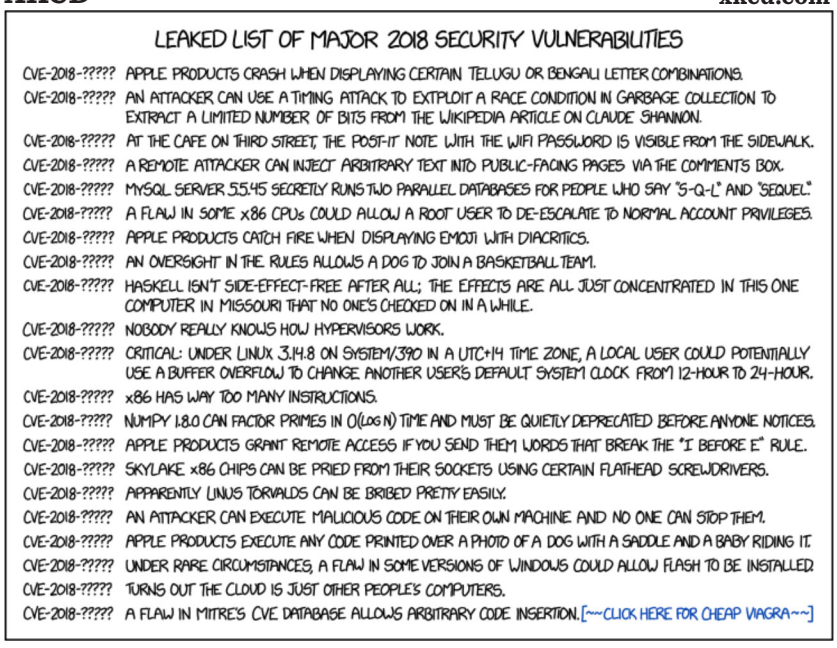
TM: We are continuing active development on the Repokid [3] project. As we add more data sources, we'll gain both more confidence in the policy suggestions and the ability to trim unused permissions even further. Specifically, we're excited about using data to constrain policy access to resources. We're starting with S3 but look forward to protecting other resources as well. Another data source that might be interesting is the software we're deploying itself. By examining the package, we may be able to infer what access it needs and remove access that it doesn't. As always, we welcome contributions to Repokid and feedback from organizations that are using it.

References

[1] J. Saltzer, M. Schroeder, "The Protection of Information in Computer Systems," in Fourth ACM Symposium on Operating System Principles (October 1973): http://www.cs.virginia.edu/~evans/cs551/saltzer/.

[2] T. McPeak, "Least Privilege: Security Gain without Developer Pain," ENIGMA Conference (USENIX, 2018): https://www.usenix.org/conference/enigma2018/presentation/mcpeak.

[3] Repokid: https://github.com/Netflix/repokid.



ENIGMA

A USENIX CONFERENCE

Submit a Talk

Enigma centers on a single track of engaging talks covering a wide range of topics in security and privacy. Our goal is to clearly explain emerging threats and defenses in the growing intersection of society and technology, and to foster an intelligent and informed conversation within the community and the world. We view diversity as a key enabler for this goal and actively work to ensure that the Enigma community encourages and welcomes participation from all employment sectors, racial and ethnic backgrounds, nationalities, and genders.

Enigma is committed to fostering an open, collaborative, and respectful environment. Enigma and USENIX are also dedicated to open science and open conversations, and all talk media is available to the public after the conference.

PROGRAM CO-CHAIRS

Ben Adida
Clever

Franziska Roesner,
University of Washington

The submission deadline is August 22, 2018.

enigma.usenix.org

JAN 28–30, 2019

BURLINGAME, CA, USA

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

Interview with Swami Sundararaman

RIK FARROW

Swaminathan (Swami) Sundararaman is the Lead Architect of ParallelM, an early-stage startup focused on production machine learning and deep learning. Swami was previously at Fusion-io Inc. and Sandisk Corp. He holds a PhD from the University of Wisconsin-Madison. swaminathan.sundararaman@parallelmachines.com

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

When I read the announcement for the HotEdge workshop [1], I was immediately intrigued. What the heck is HotEdge? I thought the "edge" consisted of network devices and content distribution networks. As I read the prospectus, I learned that edge, in this context, means something quite different from what I (and most of my friends) considered it to be.

I'll let the words of one of the co-chairs, Swami Sundararaman, do the explaining.

Rik Farrow: I thought the "edge" consisted of network devices and CDNs. But the edge in HotEdge is something different.

Swami Sundararaman: Edge computing has many definitions depending on who you ask. The one that I like the best is the following: edge computing is a new computing paradigm where server resources, ranging from a miniature computer (such as Raspberry Pi) to a small datacenter, are placed closer to data and information generation sources. Application and systems developers could use these resources to enable a new class of latency and bandwidth-sensitive applications (such as augmented reality, wearable cognitive assistance, sensor data processing, etc.) that are not realizable with current cloud computing architectures.

In most ways, edge computing is the opposite of cloud computing and therefore requires rethinking many tradeoffs that have become normal in cloud computing. Compared to its potential and the dire need for solutions for upcoming applications, we did not see workshops to foster early-stage ideas in this field as it deserves. As co-chairs, Irfan Ahmad and I wanted to help advance the field of edge computing by providing a venue where both researchers and practitioners could come together to both share their vision for building edge computing applications and systems and also discuss their nascent ideas and receive feedback well in advance of rigorous academic or industrial product treatments.

RF: So what's changed that has created the need for this new computing paradigm?

SS: There are two major trends that are driving the demand for low-latency offloading infrastructure. First, with the advent of Internet of Things (IoT), there are many more connected devices that are constantly generating tons of data (such as video, audio, sensor data, image, text, etc.). Second, the need to act or react quickly to changes provides tremendous value for businesses using sophisticated techniques (such as machine learning, deep learning, and image processing) that are both compute and memory intensive. Unfortunately, having heavy compute and/or memory demands on these sensor or other devices is not always possible for multiple reasons: power requirements, form factor, cost, development effort, etc.

Also, the latency required to move the data from these devices to the cloud (which could take multiple hops) is high and would result in a poor user experience. Even cloud providers such as Amazon and Microsoft have identified the above-mentioned trends and have started to heavily invest in edge computing (see Greengrass [2] and Azure IoT Edge [3]).

RF: Looking at your background [4], I see recent work with non-volatile memory, and older work in operating systems and storage. To me, edge computing seems very different from

what you have done in the past. What drew you to create a gathering place for edge researchers, given that you have focused on other areas in your past?

SS: This is a great question.

I love working on cutting-edge technologies and was fortunate to work on many diverse interesting problems in the past. As you have correctly observed, I started with file and storage systems research and then worked on operating systems and distributed systems in addition to traditional storage systems during my PhD. When I graduated, I jumped at the opportunity to work on non-volatile memory (including flash and persistent memory technologies), the latest upcoming storage technology at that time. At my current job, I am working on automating the deployment, orchestration, and management of machine learning in production.

As a company, we were initially interested in deploying machine learning at scale in the context of IoT. Very soon we realized that the biggest challenge to deploying performant machine learning on (or near) the "things" in IoT is the lack of infrastructure and standards. We explored the possibility of leveraging edge computing to solve our problem since it was very promising and had the potential of being the vehicle to deploy machine learning (and other upcoming compute-intensive technologies) because it addressed many of the issues (such as latency, power, compute, scale, etc.). Unfortunately, we couldn't fully embrace edge computing for its lack of wide-scale adoption. On further investigation, we discovered that there are still many open problems in edge computing and also that edge computing itself is not yet well defined.

These problems motivated me to contribute and help advance the field of edge computing. We realized that there were a few full-fledged conferences (such as SEC and Edge), but there were no workshops to discuss nascent ideas similar to what we have in other fields (such as HotOS, HotCloud, HotMobile, HotStorage, etc.). This was the primary motivation for starting HotEdge (which serves as a gathering place for edge computing researchers).

RF: Edge sounds both very interesting and important when moving forward with many technologies. But the requirements you mention, such as the need for systems that can provide enough compute power or can be scaled out to do this, will also be attractive targets. The data processed on these systems will also need to be protected. While it's still early days for edge, are people starting to think about the security requirements for these systems? I find myself imagining edge systems mining digital currency or used for spying on people using augmented reality.

SS: Yes, researchers and also industry folks have already started thinking about both security and privacy in edge computing. This is one of the key pieces needed for the adoption of edge computing as multiple entities/users could be sharing the same edge infrastructure (including CPU, memory, network, and storage). There have already been a few blogs and papers that focused on addressing both security and privacy issues in the context of edge computing.

References

[1] HotEdge Workshop: https://www.usenix.org/conference/hotedge18.

[2] Amazon Greengrass: https://aws.amazon.com/greengrass.

[3] Microsoft Azure IoT Edge: https://azure.microsoft.com/en-us/resources/videos/microsoft-ignite-2017-enable-edge-computing-with-azure-iot-edge.

[4] Swami Sundararaman at University of Wisconsin: http://pages.cs.wisc.edu/~swami/.

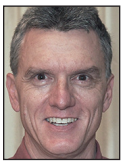
Eusocial Storage Devices

Offloading Data Management to Storage Devices that Can Act Collectively

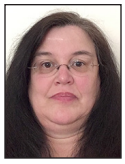
PHILIP KUFELDT, CARLOS MALTZAHN, TIM FELDMAN, CHRISTINE GREEN, GRANT MACKEY, AND SHINGO TANAKA

Philip Kufeldt is a Director of Storage Standards at Huawei Technologies, focusing on NVMe, new key value, and smart media-based storage standards. He has over 20 years of experience in storage, software, and systems. Before Huawei, Philip was at Toshiba, creating new smart media devices; Marvell, Parascale, VERITAS Software, Sun Microsystems, and IBM; and he has founded several storage-oriented startups. pak@protium.com

Carlos Maltzahn is an Adjunct Professor of Computer Science at UC Santa Cruz and the Director of the Center for Research in Open Source Software (CROSS) and the Systems Research Lab (SRL). Carlos graduated with a PhD in computer science from the University of Colorado at Boulder. carlosm@ucsc.edu



Tim Feldman works on drive design at Seagate Technology's Longmont, Colorado, Design Center. His current work focuses on object storage. He also spends time randonneuring, Nordic skiing, and logging. timothy.r.feldman@seagate.com



Christine Green led the Kinetic Open Source Project development at Seagate and continues work on Seagate's ActiveDrive™ technology. Her HDD background includes experience with recording heads and media as well as VLSI and signal processing. She has a BS in electrical engineering from Stanford University. christine.green@seagate.com

As storage devices get faster, data management tasks rob the host of CPU cycles and DDR bandwidth. In this article, we examine a new interface to storage devices that can leverage existing and new CPU and DRAM resources to take over data management tasks like availability, recovery, and migrations. This new interface provides a roadmap for device-to-device interactions and more powerful storage devices capable of providing in-store compute services that can dramatically improve performance. We call such storage devices "eusocial" because we are inspired by eusocial insects like ants, termites, and bees, which as individuals are primitive but collectively accomplish amazing things.

The Evolution of the Problem

Why Try Smart Storage Again, and Why Now?

Offloading storage processing has been around since the earliest days of computing. The idea of having a dedicated and cheaper I/O processor offloading the main processor complex made sense at a time when processor cycles were incredibly scarce and costly. However, over the years, processor cycle availability has geometrically increased and costs have plummeted making the utilitarian I/O processor costlier in terms of complexity in both hardware architecture and software. These fast and large CPU complexes permit general-purpose execution and I/O management, including data management. Data management tasks are beyond the basic tasks of storing and retrieving data, including services such as translation, mapping, deduplication, compaction, sorting, scrubbing, data movement, data redundancy, and recovery.

Including I/O management created a tight coupling of storage with the server system architecture. With such compute resources available, storage devices need only do the media management and map logical placement information to physical placement information, leaving essentially all data management relegated to the general-purpose processor. Furthermore, the simplistic API required to accomplish these goals treats every device as completely independent even though data management necessarily creates device relationships and dependencies, all of which have to be managed by the general-purpose processor.

This has driven the evolution of the storage component towards a highly cost-efficient model that has resisted most attempts to offload tasks to the component. Attempts to push some of data management back into the device, such as SCSI OSD or Kinetic, have all failed due to the need for additional compute and memory in the device pushing up per-GiB costs.

NAS Succeeds in Offloading

The one place where data management offloading was successful was Network Attached Storage (NAS). NAS environments offload all the data management to centralized servers on the network (Figure 1).

Client servers then use a network-based access protocol (NFS, CIFS) to store and retrieve data. The reason for the offloading was two-fold:

Grant Mackey works as a researcher in the office of the CTO at Western Digital Corporation in Irvine, California. His current work focuses on modeling and simulation of data-centric computing. His off time is spent hiking, cooking, and fiddling with IoT devices at home. grant.mackey@wdc.com

Shingo Tanaka is working on new types of SSD projects in the Flash Storage Department, SSD Division, Toshiba Memory Corporation, Tokyo, Japan. His current work focuses on higher functioning SSD, which can offload host tasks into SSD, effectively integrating them into existing SSD architecture. shingo3.tanaka@toshiba.co.jp

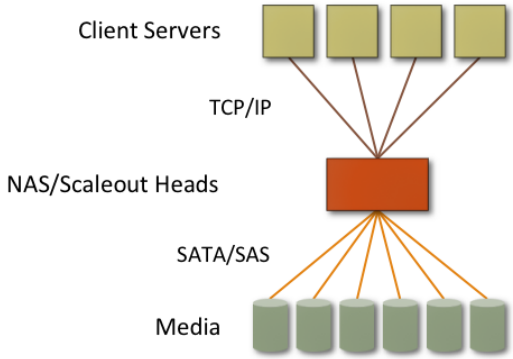


Figure 1: NAS provides centralized storage management while disaggregating storage from the server.

1. Centralizing storage management
2. Disaggregating the storage from the server

The former is a straightforward centralization gain; by consolidating all of the server management touch points, management man hours were reduced. Disaggregation was driven by the need for sharing and availability. By separating data resources from the local server and its processor complex, these resources could be placed on a general-purposes network. This provided two big wins: the data could be shared by many servers, and data resources could remain available even with the loss of the local server. This did introduce additional costs into the data layer. It was made cost effective by scaling up the number of storage devices and, hence, the number of GiB available attached to a NAS server, driving down the per-GiB costs. Also, being a central resource, this cost was further diluted by the increased number of servers being served. Even with the gains, the NAS servers themselves were tightly coupled to the storage, which created scaling limits and vulnerable islands of storage.

So, what has changed?

1. The commodity smartphone market over the last 10 years has driven the cost of embedded multi-core 64-bit processors, such as ARM, way below the cost of server processors.
2. The smartphone market also drove the power consumption of these embedded processors way down.
3. The densities of storage devices continue to skyrocket, making it easier to hide additional computing resources in the per-GiB cost.
4. New denser flash media is permitting the bandwidth aggregation of many discrete flash chips, making a single device capable of streaming GiB/s of throughput, and they will continue to get faster. This new performance level demands greater processor capabilities, and, as such, the processor costs are already partially priced into the per-GiB cost of flash devices.

But there is more. High-speed flash devices must be connected to the processor complex. Luckily, PCIe bandwidth has kept pace, growing rapidly with PCIe v3/v4, and PCIe v5 is on the horizon. However, the same cannot be said about the ultimate destination of data-system memory (Figure 2). System memory bandwidth is growing at a much slower pace than the flash devices and their interconnects. Since all I/O requests must ultimately be transferred into system memory, the system memory is already becoming the next real bottleneck.

As the storage devices deliver higher and higher throughput, the system memory bottleneck will reduce the number of storage devices that a server can effectively utilize. This problem requires that the data transferred by the server to the storage be classified and prioritized.

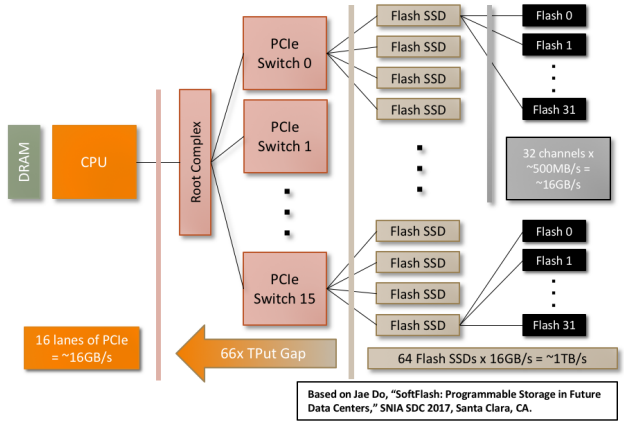


Figure 2: Throughput mismatch, based on Jae Do, "SoftFlash: Programmable Storage in Future Data Centers," SNIA SDC 2017, Santa Clara, CA

Data transfers strictly for data management (mapping/placement, scrubbing, redundancy, recovery, and accessibility) is of less benefit than actual I/O transfers for real work and should be offload targets.

RocksDB as an Example

Take the example of a key-value store like RocksDB, a library that allows an application to maintain a key-value database. The real work by the application is not likely to be the storing and retrieving of data; rather, the application is dependent on being able to persistently put and get data to/from the RocksDB store. This means any I/O transfers done to map the data and ensure its durability, availability, or accessibility is work done outside of the knowledge of the application.

RocksDB maps the key-value data through a data structure called a log-structured merge-tree (LSM). This LSM tree is implemented atop a file system, which in turn uses a block device to persistently store and retrieve data (Figure 3). The LSM tree itself constantly sorts the data as it comes into the store. This requires that large sections of data be read into the server memory via the file system and block device, manipulated and then stored into new file system structures. In addition, RocksDB ensures durability by checksumming the data as it is added to the store. It then periodically scrubs the data by transferring it to server memory and validating the checksums. All of this I/O can be considered north-south data transfers, moving data in and out of the storage device. RocksDB is not only consuming processor cycles in the sorting and scrubbing of data but, more importantly, is consuming memory bandwidth for its own data management outside of the application's knowledge.

This example gets worse when considering availability, recovery, and accessibility. In this example, RocksDB is using local storage resources. To guarantee availability of the store even if the

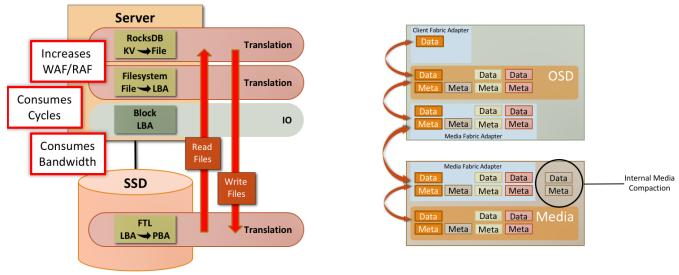


Figure 3: Translations for data management and DMA use for data management. WAF/RAF stands for the write/read amplification in Flash devices.

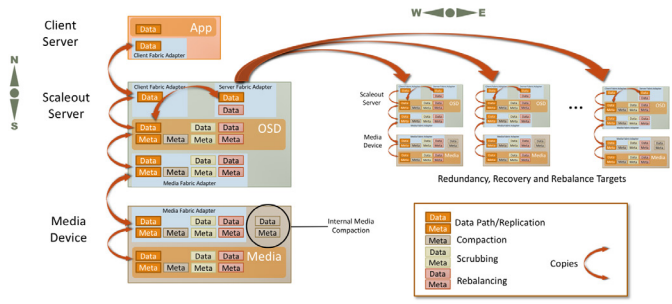


Figure 4: When scaling out a key-value database, data moves in two different dimensions: within the scale-out server (north-south) and between servers (east-west).

server fails, the data would need to be replicated to other servers with storage resources. This means data has to be transferred not only to local storage controllers but also to network controllers, greatly increasing the memory bandwidth usage. These transfers can be considered east-west transfers because the data moves laterally from one server to another (Figure 4).

Recovery again potentially moves data east and west when there are failures. Data accessibility incurs east and west data movement for the purposes of tiering, caching, or load balancing across a set of servers.

All of these activities increase the usage of the server memory bus for data management, putting the data management directly in contention with the application. With the advent of the cheap, low-power embedded processors, high-density storage devices, and high-speed devices, it is now time again to look at offloading data management to the devices themselves.

Goals of the Solution

The current standard storage API characterizes a storage device's available space as a static, linear address space of contiguous fixed-size data blocks. These address spaces can be segmented (partitioned) but have the same properties as the parent address space. Data within these address spaces is accessed in block granularities by giving a location address (logical block address, LBA) within the address space and the number

of sequential blocks to be transferred. This interface requires applications to locate their data by remembering the device, the partition, the LBA, and the length.

This location-centric model provides no other alternatives for abstract data location, data layout, redundancy, deduplication, sorting, scrubbing, data movement, data recovery, QoS, etc. Therefore, offloading data management to a storage device is more than expansion of the current storage device API—it is indeed a complete sea change.

It is important to understand the scope and high-level goals of a new storage API. The goals are:

1. Data placement within a device should be abstracted from the I/O path. Consequently, data layout should be opaque.
2. Data location should be abstracted from the I/O path.
3. Data movement from one device to another should be abstracted from the I/O path.
4. Data availability should be configurable and abstracted from the I/O path.
5. Data recovery and repair should be abstracted from the I/O path.
6. Data attributes should be supported.
7. Data access at scale should be supported.
8. Design should be mechanism-based, leaving policy to be defined by the user.

Introducing Eusocial Storage

Eusocial storage is a new API definition that drives data management activities into the device and sets a course towards in-store compute functionality. It takes into account today's scale requirements and builds on top of them.

Software

Eusocial storage is a mechanism-based software abstraction that standardizes a network/fabric-based object protocol that supports variable-sized keys and objects (Figure 5). Eusocial

- inherently disaggregates, permitting composable systems;
- inherently abstracts data location, permitting dynamic systems like scale-out storage with data availability, scaled access, and dynamic balancing;
- inherently reduces failure domains;
- supports peer-to-peer interactions, permitting autonomous data availability and data migrations between devices;
- supports device class organization, permitting scaling on a class-of-service basis;
- supports user-defined but autonomous data migrations between classes, providing for user-defined tiering and caching;
- supports in-store computing.

Hardware

Eusocial places no hard requirements on the hardware other than it must support a bidirectional network or fabric to satisfy the disaggregation, peer to peer, cluster, and control requirements. Other than this network requirement, the hardware can be defined in any fashion and take any form. There are no restrictions on media type, form factor, capacity, components, and fabric type. For example, a eusocial storage device could be a small Ethernet-enabled SSD, a small sled of HDDs, an optical jukebox, or even a media-less gateway to S3. It is anticipated that manufacturers will compete on designing hardware that is highly optimized for the media type or the targeted class of service.

Organization

Eusocial is organized into levels: storage devices, castes, and the cluster (Figure 6). The storage devices represent highly optimized, autonomous units of object-based storage. These devices define the lines of service they provide, such as throughput, latency, media type, and in-store compute availability. Devices that have similar lines of service can be organized into castes. Within a caste, devices scale out a line of service providing for data availability, data accessibility, and potentially in-store

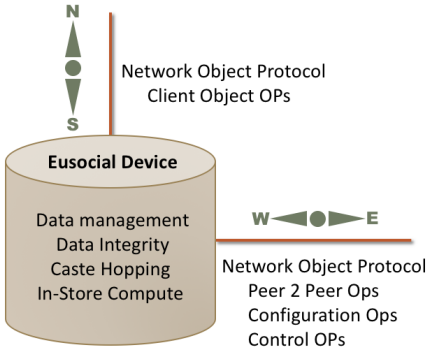


Figure 5: Eusocial device

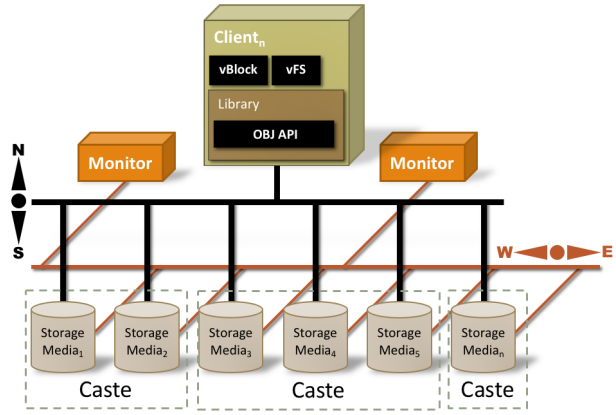


Figure 6: Eusocial hierarchy

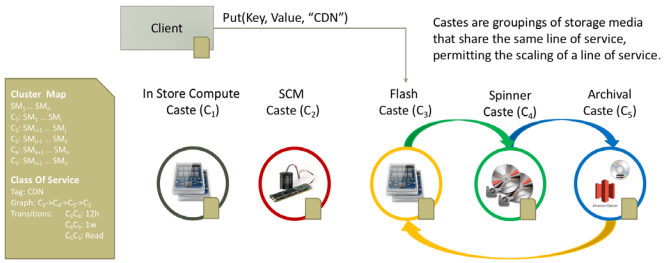


Figure 7: Eusocial caste hopping

compute. As an example, one can imagine having the following storage-media groupings:

- High-speed: a caste consisting of scaled out replicated enterprise eusocial SSDs
- Warm: a caste consisting of scaled out replicated eusocial HDDs
- Cold: a caste consisting of scaled out erasure-encoded and spin controlled eusocial HDDs
- S3: a caste consisting of a gateway to S3
- Compute: a caste consisting of scaled out in-store computing-enabled eusocial devices

Once the devices are organized into castes, users can define the lifecycle of an object by defining caste relationships, called caste maps (Figure 7). These maps plot the trajectory of an object through a set of castes and what events trigger objects to move. Once a map is defined by a user, applications can tag objects with the appropriate map. Eusocial devices themselves are responsible for following these maps and moving the data as dictated by the map.

Ultimately, the eusocial devices and castes exist inside a cluster that shares the whole configuration with all members. The cluster is also responsible for managing events and event notification. Clients also receive the configuration so that they can get and put data within the system.

Because eusocial storage is a scale-out object protocol, traditional access methods such as block and file access would be implemented atop the eusocial protocol.

The Evolution of the Solution

Changing the API is a significant issue since the block storage APIs have been ingrained in our programming model for decades. Consequently, all server software has been written to the block interface. Changing this interface will require time and some strategy to occur.

The good news is that there are significant numbers of applications that use placement abstraction storage APIs such as key-value or object. Today, these applications require a layer(s) of software like a file system to map the abstracted data to the block interface. Removing these mapping or translation layers can provide not only performance enhancements to the app but also can return processor cycles and DMA bandwidth back to the server. Paying close attention to these applications' requirements creates a ready-made set of consumers for the new API.

In addition to picking the right first-use cases, care needs to be taken on how to roll out the API's inherent complexity. An API roadmap can be broken down into several discrete steps:

1. North-south data management offloading
2. Disaggregation
3. East-west data management offloading
4. In-store computing

North-South Data Management Offloading

A natural starting point for data management offloading is happening in the industry today. There are standards bodies already working on creating a simple key-value command set that will provide an alternative to the standard block command set. This work introduces the notion of an API that has abstracted placement information behind a key-value interface and places the work of maintaining the key value store inside the device. Although initially targeted for direct connect devices, this work is being done so that it can be easily used with disaggregated protocols as well. This step begins moving applications away from the block interface programming model and onto a key-value-based interface.

The initial industry-based work will target those ready-made consumers who already use key-value APIs but have to depend on server software to implement key-value store. This effort will provide a proof point and beachhead for the eusocial API work.

Eusocial will build on this by completely providing a full object API between a eusocial device and a host. Initially this can be done on directly connected devices.

Disaggregated Storage

Many and diverse solutions prove that disaggregation is beneficial to storage workloads: NAS, Ceph, Swift, Gluster, etc. Hence the eusocial approach is revolutionary not because it is arguing for disaggregation. The novelty is the granularity of that disaggregation is now properly attributed to singularly capable devices rather than a singular server (potentially far over-scoped) responsible for a collection of dependents. So instead of having to fan-in clients only to fan-out to media (Figure 1), the eusocial approach constructs a virtual crossbar allowing clients to talk directly to all the storage devices (Figure 8).

The resulting shift increases the number of devices that need to be managed by some type of service, and that can be seen as a detriment to eusocial storage. However, we argue that this added

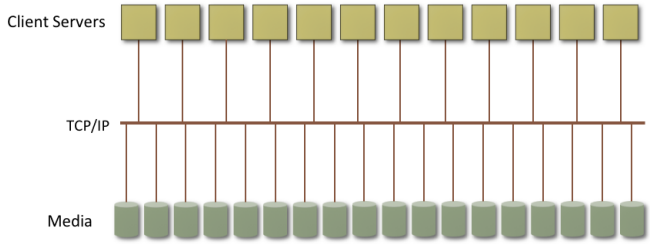


Figure 8: Full crossbar disaggregation

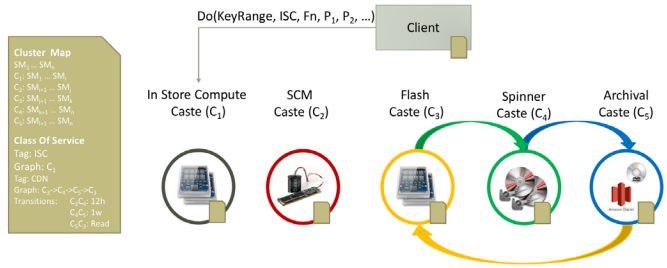


Figure 9: In-store compute caste

complexity is just added flexibility. A software-defined storage layer with many options can make better choices for applications that need a certain class of storage with a certain level of quality of service.

Eusocial storage can take advantage of existing tools such as software-defined networking to centralize management of data path and management into simple interfaces, while maintaining storage disaggregation. As an example, we have a set of eusocial castes which are connected via a software-defined, fully connected crossbar, meaning that the latency of requests from any device in any caste to any other device is similar. The actual physical architecture connecting all of these devices may differ, but the way they are advertised to client applications is this simple crossbar.

This means two things to two different groups of individuals. First, the application team enjoys a remarkable degree of freedom in choosing the type of eusocial caste their data should live and act in, without having to consider the underlying system architecture. Second, because the underlying architecture is obfuscated from the application by this layer of software-defined networking, the system infrastructure group that maintains the various devices participating in the eusocial castes has freedom in architecting the disaggregation of devices so long as they do not violate the higher level QoS guarantees being advertised by a particular eusocial caste to applications. Combined, these two groups of people are happy, and the underlying infrastructure is more efficiently consumed.

East-West Data Management Offloading

Once eusocial devices are disaggregated, the balance of the data management features of eusocial can be delivered. This includes scale access, data movement, data redundancy, data recovery, and caste hopping. These features all require peer-to-peer operations, or east-west data movement.

In-Store Computing

The design of eusocial storage naturally progresses towards "in-store computing," that is, performing computing such as data filtering, transformation, and even more compute-intensive analytics within a storage device (Figure 9). The evolution observed so far shows a slow increase of data management offloading onto storage devices, slowly increasing the requirement for processing in the device (e.g., translation layers in flash and shingled magnetic recording disks). East-west communication among in-store computing devices will enable functionalities such as deduplication, secret sharing, and divergent replication (i.e., each replica has a different layout). We anticipate that due to the increase in cost, in-store computing devices will be separate from the main storage and reside in their own caste.

Because scale out is done within the caste, implementers are free to determine the number of in-store computing eusocial devices, how they are replicated, and, through the use of caste maps, when old data should be moved out. Datacenter uses will focus around big-data processing and search. A caste of IoT eusocial devices deployed at the edge store acquired data and then do first-pass processing before shuttling the data back to the home office castes.

The benefits from in-store computing have been studied by Seagate. While many details could not be made available for this article, some of the results are promising: using benchmarks that include MapReduce, search, and data maintenance tasks, Seagate was able to double storage throughput and reduce host CPU utilization by 15–20% by offloading these tasks to devices capable of in-store computing. Seagate also found evidence that in-store computing can increase uploading speeds. In one case, uploading speeds increased by a factor of 10 over a Hadoop installation with traditional devices. All these results are mainly due to scale-out effects of offloading data-intensive operations to many devices where the data already resides and where data transfers to hosts become unnecessary.

Conclusion

This paper has examined the effects of rapidly growing storage throughput on our computing environments as well as the need for scale environments. Both of these issues are forcing a dramatic change in the roles and responsibilities of components in our systems. Previously, it was desirable to have dumb and cheap devices that a system could program and manage. But as

their capacities and speeds grow, it is becoming clear that their management and some data processing belong to the devices themselves. The eusocial concept is a design space that is opening the door to just such a future.

Acknowledgments

This work has been made possible by the Center for Research in Open Source Software at UC Santa Cruz (cross.ucsc.edu), which is funded by a donation from Sage Weil and industry memberships. Industry members include Toshiba Memory America, Inc., Micron Technology, Inc., Seagate Technology LLC, Western Digital Corporation, and Huawei Technologies Co. Ltd. CROSS, founded in September 2015 as a forum for high-impact research with strong industry participation, leverages technology transfer and standardization effects of open-source software communities and educates the next-generation of open-source software leadership among UC Santa Cruz doctoral students.

The setup for the in-store active-disk results are Linux; PLX PCIe switch, PCIe Gen-1, x4; 2 SSD drives Attribution: Nitin Kabra, Rajesh Bhagwat, Sneha Wagh; Seagate.

References

A. Acharya, M. Uysal, and J. Saltz, "Active Disks: Programming Model, Algorithms and Evaluation," *ACM SIGPLAN Notices*, vol. 33, no. 11 (1998), pp. 81–91: http://pages.cs.wisc.edu/~remzi/BrainClass/Wiki/Readings/Systems/p81-acharya.pdf.

E. Riedel, G. A. Gibson, and C. Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedia," in *Proceedings of the 24th International Conference on Very Large Data Bases*, 1998, pp. 62–73: http://www.vldb.org/conf/1998/p062.pdf.

K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A Case for Intelligent Disks (IDISKs)," *SIGMOD Record*, vol. 27, no. 3 (1998), pp. 42–52: http://db.cs.berkeley.edu/papers/sigmodr98-idisk.pdf.

H. Lim, V. Kapoor, C. Wighe, and D. H. Du, "Active Disk File System: A Distributed, Scalable File System," in *Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies*, 2001: https://pdfs.semanticscholar.org/b290/7b9e230a68250781ff4779585b2dc2a144d0.pdf.

N. Golpayegani, S. Prathapan, M. Halem, R. Warmka, B. Wyatt, J. Trantham, and C. Markey, "Bringing MapReduce Closer to Data with Active Drives," Abstract IN21D-0059 presented at 2017 Fall Meeting, AGU.

Fail-Slow at Scale

Evidence of Hardware Performance Faults in Large Production Systems

HARYADI S. GUNAWI, RIZA O. SUMINTO, RUSSELL SEARS, SWAMINATHAN SUNDARARAMAN, XING LIN, AND ROBERT RICCI

Haryadi Gunawi is a Neubauer Family Assistant Professor in the Department of Computer Science at the University of Chicago where he leads the UCARE Lab (U Chicago systems research on Availability, Reliability, and Efficiency). He received his PhD from the University of Wisconsin—Madison and was awarded an Honorable Mention for the 2009 ACM Doctoral Dissertation Award. haryadi@cs.uchicago.edu

Riza Suminto received a BS in computer science from Gadjah Mada University in 2010. In 2013, he joined the University of Chicago to pursue his PhD in computer science. He is currently a member of the UCARE Lab and is interested in addressing performance and outage bugs in cloud systems. riza@cs.uchicago.edu

Russell Sears is a Senior Engineer at Pure Storage. His research interests include high-performance and scalable systems with a focus on storage infrastructure. He is currently working on new storage APIs to replace flash translation layers and the block device abstraction. sears@purestorage.com

Swaminathan (Swami) Sundararaman is the Lead Architect of ParallelM, an early-stage startup focused on production machine learning and deep learning. Swami was previously at Fusion-io Inc. and Sandisk Corp. He holds a PhD from the University of Wisconsin—Madison. swaminathan.sundararaman@parallelmachines.com

Understanding fault models is an important criterion for building robust systems. Decades of research have developed mature failure models such as fail-stop [10], fail-partial [2], fail-transient [9], and Byzantine failures [5]. We highlight an under-studied "new" failure type: fail-slow hardware, i.e., hardware that is still running and functional but in a degraded mode, i.e., slower than its expected performance. We found that all major hardware components can exhibit fail-slow faults. For example, disk throughput can drop by three orders of magnitude to 100 KB/s due to vibration; CPUs can unexpectedly run at half-speed due to lack of power; and network card performance can collapse to Kbps level due to buffer corruption and retransmission.

While fail-slow hardware arguably did not surface frequently in the past, in today's systems, deployed at scale along with many intricacies of large-scale operational conditions, the probability that a fail-slow hardware incident can occur increases. Furthermore, as hardware technology continues to scale (smaller and more complex), today's hardware development and manufacturing will only exacerbate the problem.

To fill the void of strong evidence of hardware performance faults in the field, we—a group of researchers, engineers, and operators of large-scale datacenter systems across 12 institutions—decided to write this "community paper" [11]. More specifically, we have collected 101 detailed reports of fail-slow hardware behaviors, including the hardware types, root causes, symptoms, and impacts to high-level software.

Methodology

We collected 101 reports of fail-slow hardware from large-scale cluster deployments in 12 institutions (Table 1). At such scales, hardware is more likely to witness fail-slow occurrences. The reports were all unformatted text, written by the engineers and operators who still vividly remember the incidents due to the severity of the impacts. The incidents were reported between 2000 and 2017, with only 30 reports predating 2010. Each institution reported a unique set of root causes. For example, although an institution may have seen a corrupt buffer as the root cause slowing down networking hardware (packet loss and retransmission) many times, it was only collected as one report. Thus, a single report can represent multiple instances of an incident. If multiple institutions report the same root cause,

Institution	Nodes
Company 1	>10,000
Company 2	150
Company 3	100
Company 4	>1,000
Company 5	>10,000
University A	300
University B	>100
University C	>1,000
University D	500
Nat'l Lab X	>1,000
Nat'l Lab Y	>10,000
Nat'l Lab Z	>10,000

Table 1: Operational scale

Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems

Xing Lin is a member of the technical staff in the NetApp Advanced Technology Group. He joined NetApp after receiving his PhD from the University of Utah, where his research focused on improving space efficiency for storage systems. xing.lin@netapp.com

Robert Ricci is a Research Associate Professor in the School of Computing at the University of Utah and is one of the directors of the Flux Research Group. He has been a designer and implementer of research infrastructure for nearly two decades, including at Emulab and CloudLab. ricci@cs.utah.edu

however, it is counted multiple times. The majority of root causes (66%) were unique, however, and only 22% were duplicates (12% of the reports did not pinpoint a root cause). More specifically, a duplicated incident was reported on average by 2.4 institutions; for example, firmware bugs were reported from five institutions, driver bugs from three institutions, and the remaining issues from two institutions. The raw (partial) data set can be downloaded on our group website [1].

We note that there are no analyzable hardware-level performance logs (more in the To Vendors section, below), which prevents large-scale log studies. We strongly believe that there were many more cases that slipped by unnoticed. Some occurrences undoubtedly went unreported since operators change jobs. We did not include known slowdowns (e.g., random I/Os causing slow disks, or GC activities occasionally slowing down SSDs). We only include reports of unexpected degradation. For example, unexpected hardware faults that make GC activities work harder are reported.

Important Findings and Observations
Varying root causes: Fail-slow hardware can be induced by internal causes such as firmware bugs or device errors/wear-outs as well as external factors such as configuration, environment, temperature, and power issues.
Faults convert from one form to another: Fail-stop, -partial, and -transient faults can convert to fail-slow faults (e.g., the overhead of frequent error masking of corrupt data can lead to performance degradation).
Varying symptoms: Fail-slow behavior can exhibit a permanent slowdown, transient slowdown (up-and-down performance), partial slowdown (degradation of sub-components), and transient stop (e.g., occasional reboots).
A long chain of root causes: Fail-slow hardware can be induced by a long chain of causes (e.g., a fan stopped working, making other fans run at maximal speeds, causing heavy vibration that degraded the disk performance).
Cascading impacts: A fail-slow hardware can collapse the entire cluster performance; for example, a degraded NIC made many jobs lock task slots/containers in healthy machines, hence new jobs cannot find enough free slots.
Rare but deadly (long time to detect): It can take hours to months to pinpoint and isolate a fail-slow hardware for many reasons (e.g., no full-stack visibility, environment conditions, cascading root causes and impacts).
Suggestions
To vendors: When error masking becomes more frequent (e.g., due to increasing internal faults), more explicit signals should be thrown rather than running with a high overhead. Device-level performance statistics should be collected and reported (e.g., via SMART) to facilitate further studies.
To operators: Thirty-nine percent of root causes are external; thus troubleshooting fail-slow hardware must be done online. Due to cascading root causes and impacts, full-stack monitoring is needed. Fail-slow root causes and impacts exhibit some correlation; thus statistical correlation techniques may be useful (with full-stack monitoring).
To systems designers: While software systems are effective in handling the fail-stop (binary) model, more research is needed to tolerate fail-slow (non-binary) behavior. System architects, designers, and developers can fault-inject their systems with all the root causes reported in this study to evaluate the robustness of their systems.

Table 2: Summary of our findings and suggestions

Hardware Types						
Root	**SSD**	**Disk**	**Mem**	**Net**	**CPU**	**Total**
Device errors	10	8	9	10	3	40
Firmware bugs	6	3	0	9	2	20
Temperature	1	3	0	2	5	11
Power	1	0	1	0	6	8
Environment	3	5	2	4	4	18
Configuration	1	1	0	2	3	7
Unknown	0	3	1	2	2	8
Total	22	23	13	29	25	112

Table 3: Root causes across hardware types. "Unknown" implies that operators could not pinpoint the root cause but simply replaced the hardware. Note that a report can have multiple root causes (e.g., environment and power/temperature issues), and thus the total (112) is larger than the 101 reports.

Observations (Take-Away Points)

Varying Root Causes

Pinpointing the root cause of a fail-slow hardware is a daunting task as it can be induced in a variety of ways, as shown in Table 3. Hardware performance fault can be caused by *internal* root causes from within the device such as *firmware issues* or *device errors/wear-outs*. For example, many individual I/Os in SSD that should only take tens of µs were throttled by exactly multiples of 250µs, as high as 2-3ms; a bad batch of SSDs stopped responding for seconds and then recovered; a disk head moved slower due to gunk that spilled from the actuator assembly and accumulated between the disk head and the platter; and a NIC driver bug caused a "very poor" throughput, and the operators had to disable TCP offload to work around the problem.

However, a perfectly working device can also be degraded by many *external* root causes such as *configuration*, *environment*, *temperature*, and *power*-related issues. Some examples of external causes are: a clogged air filter caused optics in the switch to start failing due to a high temperature, generating a high 10% packet-loss rate; a partial power supply failure meant throttling the CPUs by 50%; some nodes were running slow because other nodes in the same rack were drawing more power, causing rack power supply instability and dropping power to various parts of the rack; and faulty chassis fans surrounding nodes caused such a strong vibration that drives went into recovery mode.

Fault Conversions to Fail-Slow

Different types of faults such as fail-stop, -partial, and -transient can convert to fail-slow faults.

Fail-stop to fail-slow: Because many hardware pieces are connected, a fail-stop component can make other components exhibit a fail-slow behavior. For example, a dead power supply throttled the CPUs by 50% since the backup supply was unable to deliver enough power; and a vendor's buggy firmware made a batch of SSDs stop for seconds, disabling the flash cache layer and slowing the entire storage stack. These examples suggest that fail-slow occurrences can be correlated to other fail-stop faults in the system. A robust fail-stop-tolerant system should ensure that a fail-stop fault does not convert to fail-slow.

Fail-transient to fail-slow: In addition to fail-stop, many kinds of hardware can exhibit fail-transient errors: for example, disks occasionally return I/O errors, processors sometimes produce a wrong result, and memory corrupts from time to time. Due to their transient and "rare" nature, firmware/software typically masks these errors from users. A simple mechanism is to *retry* the operation or *repair* the error (e.g., with ECC or parity). When the transient failures are recurring much more frequently, however, *error masking* can be a "double-edged sword." That is, because error masking is not a free operation (there are retry delays, repair costs), when the errors are not rare, the masking overhead becomes the common case performance.

We observed many cases of fail-transient to fail-slow conversion. For example, a disk firmware triggered frequent "read-after-write" checks in a degraded disk; and many cases of loss/corrupt network packets (a 1–50% rate in our reports) triggered heavy retries that collapsed the network throughput by orders of magnitude.

From the stories above, it is clear that a distinction must be made between rare and frequent fail-transient faults. While it is acceptable to mask the former, the latter should be exposed to and not hidden from high-level software stack and monitoring tools.

Fail-partial to fail-slow: Some hardware can also exhibit fail-partial fault where only some part of the device is unusable (that is, a partial fail-stop). This kind of failure is typically masked by the firmware/software layer (e.g., with remapping). However, when the scale of partial failure grows, the fault masking brings a negative impact to performance. Bad chips in SSDs reduce the size of over-provisioned space, triggering more frequent garbage collection; and a more known problem, remapping a large number of bad sectors, can induce more disk seeks. Similar to the fail-transient case above, there must be a distinction between small- and large-scale partial faults.

Varying Fail-Slow Symptoms

We observed the "many faces" of fail-slow symptoms: permanent, transient, and partial fail-slow and transient fail-stop, as illustrated in Figure 1.

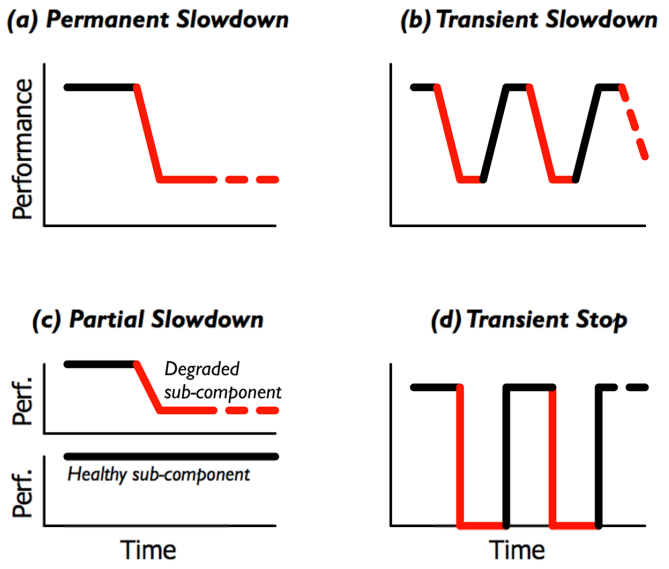


Figure 1: Fail-slow symptoms. The figure shows four types.

Permanent slowdown: The first symptom (Figure 1a) is a permanent slowdown, wherein the device initially works normally but its performance drops off over time and does not return to the normal condition (until the problem is manually fixed). This mode is the simplest among the four models because operators can consistently see the issue.

Transient slowdown: The second symptom (Figure 1b) is a transient slowdown, wherein the device's performance fluctuates between normal and significant degradation, which is more difficult to troubleshoot. For example, applications that create a massive load can cause the rack power control to deliver insufficient power to other machines (degrading their performance), but only until the power-hungry applications finish.

Partial slowdown: The third model (Figure 1c) is a partial slowdown, where only some parts of the device slow. In other words, this is the case of partial fail-stop converting to partial slowdown. For example, some parts of memory that are faulty require more ECC checks to be performed. The partial fail-slow model also complicates debugging since some operations experience the slowdown but others on the same device are not affected.

Transient stop: The last symptom (Figure 1d) is the case of transient stop, where the device occasionally reboots itself, causing performance at times to degrade to zero. For example, a buggy firmware sometimes made the SSDs "disappear" from the RAID controller and later reappear.

Cascading Causes and Impacts

Another intricacy of fail-slow hardware is the chain of cascading events: First, between the actual root cause and the hardware's fail-slow symptom, there is a chain of *cascading root causes*.

Second, the fail-slow symptom then creates *cascading impacts* to the high-level software stack, and potentially to the entire cluster.

Here are some examples of long cascading root causes that lead to fail-slow hardware. A fan in a compute node stopped working, making other fans compensate for the dead fan by operating at maximal speeds, which then caused a lot of noise and vibration that subsequently degraded the disk performance. When a piece of hardware becomes fail-slow, not only does it affect the host machine, but it can cause cascading impacts across the cluster. For example, a degraded NIC in one machine, slowing from 1 Gbps to 1 Kbps, caused a chain reaction that slowed down the entire cluster of 100 machines as the connecting tasks that were affected held up containers/slots for a long time, and new jobs could not run due to the slot shortage.

Rare but Deadly: Long Time-to-Detect

The fail-slow hardware incidents in our report took *hours* or even *months* to detect (pinpoint). More specifically, 1% of the cases were detected in minutes, 13% in hours, 13% in days, 11% in weeks, 17% in months, with an unknown time in 45% of cases. Some engineers called this a "costly debugging tail." In one incident, an entire team of engineers was pulled to debug the problem, costing the institution tens of thousands of dollars. There are several reasons why the time-to-detect (TTD) is long.

First, the fact that the incidence of fail-slow hardware is not as frequent as fail-stop cases implies that today's software systems do not completely anticipate (that is, undermine) such scenarios. Thus, while more-frequent failures can be solved quickly, less-frequent but more complex failures (that cannot be mitigated by the system) can significantly cost the engineers time.

Second, as explained before, the root cause might not originate from the fail-slow hardware. For example, the case of transient slowdown caused by power-hungry applications took months to figure out since the problem was not rooted in the slow machines nor the power supply.

Third, external-environment conditions beyond the control of the operators can prolong diagnosis. For months, a vendor failed to reproduce the fail-slow symptoms in its sea-level testing facility since the hardware only slowed down at a high mountain altitude.

Finally, operators do not always have full visibility of the entire hardware stack. For example, an incident took days to solve because the operators had no visibility into the power supply health.

Suggestions

In addition to cataloging instances of fail-slow hardware, a goal of this study is to offer vendors, operators, and systems designers insights about how to address this poorly studied failure mode.

To Vendors

Making implicit error masking explicit: Fail-slow hardware can be categorized as an "implicit" fault, meaning it does not always return any explicit hard errors (e.g., due to error masking; see the Fault Conversions to Fail-Slow section, above). However, there were many cases of slowly increasing error rates that would eventually cause cascading performance failures. Vendors might consider throwing explicit error signals when the error rates far exceed the expected rate.

Exposing device-level performance statistics: Modern hardware now exposes such information via SMART. However, our conversations with operators suggest that the information from SMART is "insufficient to act on." We hope vendors will expose device-level performance data to support future statistical studies.

To Operators

Online diagnosis: In our study, 39% of the cases were caused by external root causes. Some reports suggest that operators took days or even months to diagnose the problem since it could not be reproduced in offline testing. Thus, online diagnosis is important, but also not straightforward, because not all hardware components are typically monitored, which we will discuss next.

Monitoring of all hardware components: Today, in addition to main hardware components, other hardware components and environment conditions such as fan speeds and temperature are also monitored. Unfortunately, not all hardware is monitored in practice. For example, multiple organizations failed to monitor network cables, using the flow of traffic as a proxy for cable health instead. The diagnosis took much longer because blame for poor performance is usually directed towards the main hardware components such as NICs or switches. The challenge is then to prevent too much data being logged.

Another operational challenge is that different teams are responsible for different parts of the data center: software behavior, machine performance, cooling, power. With limited views, operators cannot fully diagnose the problem.

A future challenge relates to a proprietary full-packaged solution like hyper-converged or rack-scale design. Such design usually comes with the vendor's monitoring tools, which might not monitor and expose all information to the operators. Instead, vendors of such systems often monitor hardware health remotely, which can lead to fragmentation of monitoring infrastructure as the number of vendors increases.

Correlating full-stack information: With full-stack performance data, operators can use statistical approaches to pinpoint and isolate the root cause [6].

Although most of the cases in our study were hard-to-diagnose problems, the revealed root causes were relatively "simple." For example, when a power-hungry application was running, it drained the rack power and degraded other nodes. Such a correlation can easily be made but requires processing of power-level information. Future research can be done to evaluate whether existing statistical monitoring approaches can detect such correlations.

While the metrics above are easy to monitor, there are other fine-grained metrics that are hard to correlate. For example, in one configuration issue, only multicast network traffic was affected, and in another similar one, only big packets (>1500 bytes) experienced long latencies. In these examples, the contrast between multicast and unicast traffics and small and big packets is clear. However, to make the correlation, detailed packet characteristics must be logged as well.

Finally, monitoring algorithms should also detect "counter-intuitive" correlations. For example, when user performance degrades, operators tend to react by adding more nodes. However, there were cases where adding more nodes did not translate to better performance since the underlying root cause was not isolated.

To Systems Designers

While the previous section focuses on post-mortem remedies, this section provides some suggestions on how to better anticipate fail-slow hardware in future systems.

Making implicit error-masking explicit: Similar to hardware, error masking (as well as "tail" masking) in higher software stacks can make the problem worse. We have observed fail-slow hardware that caused many jobs to time out and be retried again repeatedly, consuming many other resources and converting the single hardware problem into larger cluster-wide failures. Software systems should not just silently work around fail-slow hardware but need to expose enough information to help troubleshooting.

Fail-slow to fail-stop: Earlier, we discussed many fault conversions to fail-slow faults. The reverse can be asked: can fail-slow faults be converted into fail-stop mode? Such a concept is appealing because modern systems are well equipped to handle fail-stop failures [3]. We next discuss opportunities and challenges of this concept.

Skip non-primary fail-slow components: Some resources, such as caching layers, can be considered nonprimary components. For example, in many deployments, SSDs are treated as a caching layer for the back-end disks. The assumption that SSD is always fast and never stalls does not always hold. Thus, when fail-slow SSDs (acting as a caching layer) introduce more latencies than the back-end disks, they can be skipped temporarily until the problem subsides. However, consistency issues must be taken into account. Another suggestion is to run in "partial"

mode rather than in full mode but with slow performance. For example, if many disks cause heavy vibration that degrades the disk throughput significantly, it is better to run fewer disks to eliminate the throughput-degrading vibration [4].

Detect fail-slow recurrences: Another method to make slow-to-stop conversion is to monitor the recurrence of fail-slow faults. For example, when disks or SSDs continue to "flip-flop" between online/offline mode, triggering RAID rebalancing all the time, it is better to take them offline. We observed several cases of transient fail-slow hardware that was taken offline, but after passing the in-office diagnosis, the device was put online again, only to cause the same problem.

Fail-slow fault injections: System architects can inject fail-slow root causes reported in this study to their systems and analyze the impacts.

One can argue that asynchronous distributed systems (eventual consistency) should naturally tolerate fail-slow behaviors. While this is true, there are many stateful systems that cannot work in fully asynchronous mode: in widely used open-sourced distributed systems, fail-slow hardware can cause cascading failures such as thread pool exhaustion, message backlogs, and out-of-memory errors [8].

Tail-tolerant distributed systems [7] are supposed to be resilient. However, other recent work shows that the "tail" concept only targets performance degradation from resource contention, which is different from the fail-slow hardware model such as slow NICs; as a result, not all tail-tolerant systems, like Hadoop or Spark, can cut tail latencies induced by degraded NICs [13].

Beyond networking components, the assumption that storage latency is stable is also fatal. It has been reported that disk delays cause race conditions or deadlock in distributed consistency protocols [12]. The problem is that some consistency protocols, while tolerating network delays, do not incorporate the possibility of disk delays, for the sake of simplicity.

With fail-slow injections, operators can also evaluate whether their systems or monitoring tools signal the right warnings or errors. There were a few cases in our reports where wrong signals were sent, causing the operators to debug only the healthy part of the system.

Overall, we strongly believe that injecting root causes reported in this study will reveal many flaws in existing systems. Furthermore, all forms of fail-slow hardware such as slow NICs, switches, disks, SSD, NVDIMM, and CPUs need to be exercised since they lead to different symptoms. The challenge is then to build future systems that enable various fail-slow behaviors to be injected easily.

Conclusion

Today's software systems are arguably robust at logging and recovering from fail-stop hardware—there is a clear, binary signal that is fairly easy to recognize and interpret. We believe fail-slow hardware is a fundamentally harder problem to solve. It is very hard to distinguish such cases from ones that are caused by software performance issues. It is also evident that many modern, advanced deployed systems do not anticipate this failure mode. We hope that our study can influence vendors, operators, and systems designers to treat fail-slow hardware as a separate class of failures and start addressing them more robustly in future systems.

Complete List of Authors

Haryadi S. Gunawi, Riza O. Suminto, Mingzhe Hao, and Huaicheng Li, University of Chicago

Russell Sears and Casey Golliher, Pure Storage

Swaminathan Sundararaman, Parallel Machines

Xing Lin and Tim Emami, NetApp

Weiguang Sheng and Nematollah Bidokhti, Huawei

Caitie McCaffrey, Twitter

Gary Grider and Parks M. Fields, Los Alamos National Laboratory

Kevin Harms and Robert B. Ross, Argonne National Laboratory

Andree Jacobson, New Mexico Consortium

Robert Ricci and Kirk Webb, University of Utah

Peter Alvaro, University of California, Santa Cruz

H. Birali Runesha, Mingzhe Hao, and Huaicheng Li, University of Chicago Research Computing Center

References

[1] Download the fail-slow database: http://ucare.cs.uchicago.edu/projects/failslow/.

[2] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," in *Proceedings of the 2007 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007: http://research.cs.wisc.edu/adsl/Publications/latent-sigmetrics07.pdf.

[3] G. Candea and A. Fox, "Crash-Only Software," in *Proceedings of the Ninth Workshop on Hot Topics in Operating Systems (HotOS IX)*, 2003.

[4] C. S. Chan, B. Pan, K. Gross, K. Gross, T. S. Rosing, and K. Vaidyanathan, "Correcting Vibration-Induced Performance Degradation in Enterprise Servers," in *Proceedings of the Greenmetrics workshop (Greenmetrics)*, 2013: http://seelab.ucsd.edu/papers/cschan_gm13.pdf.

[5] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults," in *Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI)*, 2009: https://www.usenix.org/legacy/event/nsdi09/tech/full_papers/clement/clement.pdf.

[6] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, N. Arora, and G. Jiang, "PerfScope: Practical Online Server Performance Bug Inference in Production Cloud Computing Infrastructures," in *Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC)*, 2014: http://www.nipunarora.net/pdf/perfscope.pdf.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, 2004: https://www.usenix.org/legacy/event/osdi04/tech/full_papers/dean/dean.pdf.

[8] T. Do, M. Hao, T. Leesatapornwongsa, T. Patana-anake, and Haryadi S. Gunawi, "Limplock: Understanding the Impact of Limpware on Scale-Out Cloud Systems," in *Proceedings of the 4th ACM Symposium on Cloud Computing (SoCC)*, 2013: http://ucare.cs.uchicago.edu/pdf/socc13-limplock.pdf.

[9] T. Do, T. Harter, Y. Liu, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "HardFS: Hardening HDFS with Selective and Lightweight Versioning," in *Proceedings of the 11th USENIX Symposium on File and Storage Technologies (FAST)*, 2013: https://www.usenix.org/system/files/conference/fast13/fast13-final70_0.pdf.

[10] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar, "Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages," in *Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC)*, 2016: http://ucare.cs.uchicago.edu/pdf/socc16-cos.pdf.

[11] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, "Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems," in *Proceedings of the 16th USENIX Symposium on File and Storage Technologies (FAST '16)*, 2018: https://www.usenix.org/system/files/conference/fast18/fast18-gunawi.pdf.

[12] T. Leesatapornwongsa, J. F. Lukman, S. Lu, and H. S. Gunawi, "TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016: http://ucare.cs.uchicago.edu/pdf/asplos16-TaxDC.pdf.

[13] R. O. Suminto, C. A. Stuardo, A. Clark, H. Ke, T. Leesatapornwongsa, B. Fu, D. H. Kurniawan, V. Martin, U. M. Rao G., and H. S. Gunawi, "PBSE: A Robust Path-Based Speculative Execution for Degraded-Network Tail Tolerance in Data-Parallel Frameworks," in *Proceedings of the 8th ACM Symposium on Cloud Computing (SoCC)*, 2017: http://ucare.cs.uchicago.edu/pdf/socc17-pbse.pdf.

A Quarter Century of LISA
(with Apologies to Peter Salus)

SEAN KAMATH

Sean Kamath is a Production Engineer at Facebook. He has been a lurker at LISA conferences since 1992. kamath@moltingpenguin.com

In 1992, I had just hired another sysadmin named Dan. One day he asked me about going to this conference, which I had never heard of. I'd never been to any sort of conference before and had no idea what to expect. What we found, however, was amazing. We had found our people. It seemed like everyone there was just like us, all struggling with the same things, all looking for similar solutions. In retrospect, in such a young industry, this wasn't all that surprising. Perhaps the most surprising thing, however, was the vaunted "hallway track" and learning the backgrounds of everyone I met. I don't actually recall meeting a single CS major. Science was well represented (I'm a physics major, and there were plenty of math, biology, chemistry, and physics majors), but in a strange twist there were also plenty of music, art, theater, literature, and all sorts of other liberal arts majors.

This was my first LISA conference, LISA VI, held in Long Beach, CA. You won't find it on the USENIX website (you can find a reference to it if you try hard enough). I have attended every LISA conference since then, the last being the 31st LISA, held in San Francisco. While I've written before of my experience attending LISA ("Whither LISA," *;login:* February 2010), what follows are my thoughts and observations about the conference over these 25 years rather than a history of the first 25 years. When I started in the system administration field, the sysadmin was a mysterious creature. Most companies didn't know they needed one. Most probably didn't need one. My first UNIX experience, when I went to college in 1984, was on the school's VAX running BSD4.1.

What a lot of people these days don't know is that system administration back then was very often done by one of two types of people: folks who wanted to be software developers but weren't quite there, and folks who just fell into it. People in the former camp usually only did the job for a year or two until they could get a job as a developer. The rest of us? We just did it because we could.

So this was the beginning of my experience at LISA, a conference full of people doing something they hadn't been formally trained to do but who did it because they loved it or were good at it (often both), where the goal of attending was to learn, experience, discover. Swag was nonexistent and unexpected. Vendors didn't exist. It was a conference set up and run by people attending. It was like a bunch of geeks just congregating in the same place and spending a week talking about everyone they were dealing with.

I hope I've set the stage appropriately as an attendee. However, I should take a moment to say that I know that organizing this seemingly spontaneous congregation of sysadmins actually took a lot of work, and that a lot of dedicated, smart, and talented people made it appear completely seamless. Further, I believe that over the years, as conference organizers have come and gone, it has become increasingly hard to do what they do. Having been on the papers committee, I know how incredibly hard it is to find quality papers, and I was not at all surprised when the refereed papers track went away, a victim of the internet, in my opinion.

As I look back on these last 25 years, I see a similarity to my college years. I've watched LISA start out as a freshman and develop into a senior. And, conveniently for me, I had two junior years, so that makes it a nice five years per...year. I'm reminded that no analogy is perfect.

Freshman

As I've explained, sysadmins during this time were mostly people who fell into the job. When I hired Dan (who had been at the company a long time in a different role), he admitted to me that he was unsure whether this was a direction he wanted to go in, uncertain whether system administration was something you could do for an extended period. In short, it wasn't a career path. And yet, at that first conference we attended, in one of the tutorials or talks (time clouds my memory), someone said, "It's OK to make system administration your career." Dan was so relieved to hear this. I was too young and naïve to realize it was a watershed moment until much later.

Sometimes it's hard to explain to people that this time was at the very beginning of the internet as we know it. Forget smartphones—this was before the graphical web browser (NCSA Mosaic was released in 1993). Discussions would often mention Gopher. What this meant in practical terms was that we all tended to be isolated, with no way to easily see new things being created, no easy way to discover new tools, no way to broadcast that you had written this cool thing that did something everyone needed done. It's not that there was no way (I'm looking at you, NetNews), but it took time.

In addition, so many building blocks of what we know and love today, while not exactly hot off the presses, were still pretty darn new. Everyone ran their own mail server, people were excited about using DNS—for large installations, DNS was almost a must—and might have used BOOTP (defined in an RFC from 1985), but it wouldn't be until 1997 that anyone was using DHCP. It could take years for enough people to realize their problems were similar to others', to craft some notion of how their problems might be solved, and to actually write and disseminate code.

This is what made LISA so amazing. We could all get together, those of us charged with the responsibility of running large installations, and accelerate the dissemination of these tools, practices, ideas, and experiences. One of the great things about LISA at this time was that the refereed papers were where people could publish their code. The loudest groans from the attendees would come when someone who'd written a paper on some great new tool, when asked about getting it, would reply, "We're still trying to see if we can release this."

It wasn't just the technical side where we came together and discussed things. Rob Kolstad's tutorial on the ethics of system administration opened our eyes to issues never thought of. There were sessions on being a manager. On running a team. On the human side of dealing with so many users.

In many ways, this was the Golden Age of LISA. It was achingly relevant. It was so necessary. It was everything—reckless, bawdy, outrageous, as well as thoughtful, helpful, and just plain wonderful.

Sophomore

As the profession of system administration grew and developed, things got more serious. No longer were the hotel room parties stuffed with people, with a bathtub full of beer and ice in the bathroom, the bed tipped up against the wall (my first BayLISA "hospitality suite"). Now they were stuffed with people, hosted by companies looking to hire. People, and companies, thought that much of what the sysadmin did was a "solved problem." If not solved, well, one or two software updates away from being solved. Just look at configuration management. (CFEngine vs. Bcfg2! And, yow, the Wikipedia page on CFEngine doesn't even mention Bcfg2.)

This was an age of vendors trying so hard to provide solutions for all the companies that were just starting to be aware that they needed people to run their computers, but couldn't find the people that had the experience to do the job well. No longer could you hire a wannabe software developer to run the computer. And when the sysadmin failed, real work was lost, either in lost time or lost data.

System administration was All Things Computer around this time, which meant trying to find ways of managing PCs as well as the central servers. The conference started to draw people from all sorts of shops, not just "Large Installations." The internet was starting to take off, especially for high-tech companies. This had two effects on the conference. First, it got the word out. People would put up URLs to get their code. Mailing lists cropped up, and communities formed around software. Second, and perhaps somewhat detrimental to the existing conference format, the ability to directly publish to the web meant that some folks started to release software, and then talk about it at LISA.

This period created two diametrically opposed problems for those running the conference. It saw the dot-com bubble, as well as the dot-com bust. And with that came the massive growth of the conference, followed by an implosion of attendees. Imagine one year with over 2000 attendees, and the next, fewer than 1000. I don't know the actual numbers, but I was there. One year you couldn't find a place to sit at the tutorial lunches, the next was a grim affair where we ate in silence with a lot of empty seats at all the tables. The number of companies that showed up

and plied their wares similarly imploded—not just the vendors, but the companies sending their employees. Where once multiple people would show up from major companies, there now was one person who got to go that year.

It was a grim time.

Junior

The conference had to find a way to stay relevant after the dot-com bust. Not because what it had to offer wasn't relevant anymore, far from it. All the companies out there still needed people to do their job. And now, more than ever, they needed to be as efficient as possible. Just like my (first) Junior year in college, it was time for the conference to buckle down and get to work.

The pace of innovation slowed only slightly around this time. People couldn't afford to attend the conference, but problems still needed to be solved. The dot-com bubble brought a whole new paradigm for how systems could be used and for the importance of the internet. With it came the challenges of running systems that people other than your users used.

Around this time, I think the conference reached the peak of trying to be all things to all people. Run PCs? Come to LISA. Run internet services? Come to LISA. Running corporate infrastructure? Networking? Storage? Suddenly, "All Things Computer" was simply too much. You could see it in the huge expanse of the refereed papers, invited talks, and BoFs.

The culmination of these two things (people couldn't afford to attend, and the conference trying to appeal to as broad an audience as possible to attract more attendees) resulted in both a wide variety of topics covered, as well as a similarly wide variety in quality and applicability. One thing to keep in mind about this time, the early aughts, is that even after the bubble burst there was an explosion of opportunity for companies with this internet thing, as well as for corporate systems (everyone had a computer at work, it seemed).

Additionally, while we had accepted methodologies for how to provide certain services, etc., the pace of solutions couldn't keep up with the new problems companies and universities found as they expanded. Issues of scalability, reliability, reachability, accountability, securability—well, you get the idea. These issues outstripped everyone's ability to find solutions.

The upshot of all this was that the conference tilted away from direct solutions and toward explorations of possible solutions, along with descriptions of problems encountered. The invited talks expanded from one track to two and became better attended as people looked for insights into problems they were running into. Workshops were added to the training, serving as a way to delve more deeply into problems that were moving from edge cases to commonly encountered.

Junior (Redux)

What happens when what you thought you were doing correctly turns out to need a course correction? It was in the middle of this period that I wrote "Whither LISA," my attempt to reflect on what LISA meant to me and to issue a call to action to come together and infuse new life into the conference. I felt that LISA had, to some extent, reached a crisis point. It could no longer be the conference that covered everything. There were other conferences competing for attendees, with a slightly different slant.

I think a lot of the genesis for this stems from the growth of the field. We went from system administrators to system engineers, then system/network/storage engineers, and then…site reliability engineers. Then along came DevOps.

When I think about the evolution of the role of "people who make machines work," I think of this: the very first system administrators were the people who wrote the operating systems for the simple reason that they built the OS and there was no one else to manage it. As with all things run by the people who built them, it was assumed that the person running the system knew everything the person who built it knew.

Eventually, the UNIX distributions landed at sites with people who didn't build them. And thus, people who didn't have that intimate knowledge of how things were constructed were responsible for keeping things working nonetheless. And, as often happens, those people, freed of the presumption of how things *should* work, could often make things run quite well, sometimes in unexpected ways.

As the problems faced by sites grew more diverse, complicated, and involved, the need for specialization grew. At first, the need was for base technologies (storage, networking), but eventually it grew even to encompass what the system was designed to solve. The skills and tools used to provide a reliable environment for doing software development were often dramatically different from, and yet in some ways the same as, providing a thousand office workers with functional, up-to-date PCs.

And then came the web. Suddenly we had a whole new denizen of the system administrator realm: software developers working on systems that were providing services that system administrators might support, but the support was for customers. Sometimes they even paid.

As for the companies that had *huge* environments, well, they experienced the most problems. They also had the resources (people, money, motivation) to solve the problems. And many of those companies were very involved in open source software and were more than willing to share with their communities.

The result was that there was a bit of a dichotomy at LISA. There were the people from the large companies who were facing problems that were unique to their industry but who were driving the

talks. There were also the folks from smaller shops who started to feel sidelined. I can't tell you the number of times that I heard someone talk about the lack of applicability of talks and papers to their job. When LISA first ramped up, people who were new, even those from very small shops, found a lot of useful information they could use, sometimes right away. Fifteen years later, what was someone running a small shop of 100 people and a few servers going to do with the information about running Hadoop clusters? What was I, working at a fairly large animation studio, going to do with that information? I tried to glean as much useful information from the various sources as possible. At least my environment wasn't that far off from the big guys. But I saw a lot of folks from small shops stop coming and heard a lot of people say this was their first, and likely last, LISA.

Senior

Around 20 years after I started attending LISA, after much internal upheaval and change over those intervening years, LISA started to morph. Not a lot, but just enough. Instead of everything being about concrete "solutions," instead of restatement of problems, instead of a little bit of everything, there started to be a convergence. Two things started to become apparent.

First, a thread that had been common since the beginning of LISA became mainstream: it's not about the current problems; it's about how we navigate around to find solutions. It's about the habits, behaviors, and techniques that people should develop to help them find their way in this job. Be it managing managers, budgets, time, or systems, useful information about strategies and examples of people facing and handling challenges became a fundamental part of the tutorials, invited talks, and, of course, the hallway track. As a smart friend of mine recently said: "Most of what we as sysadmins, of whatever flavor, do is to learn enough about how something works, then adapt it so that it works in our environment. We learn from examples but must be able to apply those examples." So true.

Second, everyone was an internet company now. While everyone still had the challenge of running internal systems, that turned out to be fairly static. But after 20 years of trying to figure out how to provision and configure internal systems (be they PCs or whatever), people and companies (and vendors!) knew how to do it. Even the explosive growth of mobile was addressed in short order by vendors and companies with BYOD policies and apps.

What made this go-round different from the last one was this: The Cloud. Yeah, a lot of us laughed at the term. I still do. But it turned out that companies were thrilled to offload their work onto other companies. We saw everything as a service, to the point that these days we have Services as a Service (turtles all the way down). Companies were more than happy to farm out their email to a few large vendors. Speaking as someone who still runs a small personal mail server, this freaks me out. We saw companies migrate (sometimes with good reasons) to cloud-based document storage and editing.

But the killer app turned out to be cloud-based servers. No longer did companies shell out capital for hardware to run their services. Instead, they could just rent the machines and have all the benefits of a highly paid professional staff to run them. Software to run all these machines was created, and all you had to do was learn how to use it.

And so, in a bizarre twist, the cloud became ouroboros. The companies that were providing *aaS and the customers of those companies had a convergence of interest, one solved by attending LISA. Both groups of people used the same, or similar, tools, albeit for very different ends. Sure, Ansible/Salt vs. Chef/Puppet replaced CFEngine vs. Bcfg2, as even cloud-based services need configuration management of some sort. If you needed a huge Hadoop cluster, no problem; fire up a couple thousand machines for a week or two.

Graduate School?

So where does that leave us now? Well, recently USENIX announced a significant change to the LISA conference. Gone are the dedicated three days of training. Tutorials will live on in the mini-tutorial, but the need for half a day or a full day to fully grok some key concepts is gone. People need and want self-guided training, or access to knowledgeable people to ask questions of. The conference will become smaller and more focused, I think. I don't know if smaller in size, but definitely compressed in time.

Change is hard. And for an old-timer like me, it's really scary. I will be interested in, and trepidatious about, the changes that are in store and how they will play out. I don't think this is the last change we'll see and with good reason. If the conference hadn't changed from that first one in 1992, I would have stopped going by 1995. And I have a lot of trust in the fine folks at USENIX and the people from our own community who make up the steering committee of the conference. Because one thing has remained constant despite the changes and tumult over all the years: it's still a conference by and for the people who are putting it on.

Practical Perl Tools

It's a Relationship Thing

DAVID N. BLANK-EDELMAN

David has over 30 years of experience in the systems administration/DevOps/SRE field in large multiplatform environments and is the author of the O'Reilly Otter book (new book on SRE forthcoming!). He is one of the co-founders of the now global set of SREcon conferences. David is honored to serve on the USENIX Board of Directors where he helps to organize and engineer conferences like LISA and SREcon. dnb@usenix.org

Faithful readers of this column will know I have a declared affection for graphs. In the past, we've looked at ways to represent graphs in Perl and ways to draw them. But strangely enough, we've never looked at one of the more interesting uses for them these days: as a data representation for a database, that is, graph databases. This column aims to right that wrong. Rather than take on the entirety of the graph database space in this column, we'll use Neo4j, one of the more popular ones, as a springboard for how they work and how we can interact with them via Perl.

The Things and the Other Things (the Basics)

Graph databases are all about dealing with pieces of data and the relationships between those pieces of data. But wait (you cry out in alarm), "Aren't all databases about this? Cough, cough, relational databases, cough, cough..."

(super oversimplification alert!) Relational databases store information in a number of tables containing rows of data (records). The row is broken up into columns (fields). We perform operations that attempt to find matches between the contents of a column in one table and a column in another table. When we get a match, we treat the corresponding rows as related. The usual example of this is a table of people and a table of addresses. If both tables share a "Person ID" column, we can pick an ID from the people table, match it to the address table, and determine that person's address or addresses. Let's say we want to also store orders this person has placed. Add another table. And stored payment info for this person? Add another table. Membership in a special discount group? Add another table. Cumbersome, but pretty straightforward.

To make this example more interesting, let's say we want to introduce a new concept of a family and keep track of the relationship between the people in that people table. Can do, but it starts to get more and more hairy the more complicated these relationships get. Anyone who has had to work with gnarly JOIN statements knows exactly what I am talking about. And the point where you have to redo your entire data model based on new requirements for data representation? No fun at all.

Graph databases (Neo4j in particular, to keep this concrete) go about this in a different way. In Neo4j, you have nodes. These are the things you are storing. A person, an address, an order, a piece of payment info, a discount group, that sort of thing. Each node has a label (typically used to identify the "kind" of node—examples would be *person* or *address*) and a set of properties stored in the node (examples: first/last name, street name, item number, credit card number, discount percentage).

Now let's add the relations part that will construct the graph. We can connect two nodes via a unidirectional (more on this later) relationship. For example: CHILD, MARRIED, LIVES_AT, ORDERED, JOINED. Relationships are first-class things unto themselves. They have labels like the ones I just used as examples (CHILD, MARRIED, etc.) and properties too (e.g., MARRIED could have a property of wedding_date).

If I wanted to write some of these nodes and relationships down, you could imagine I might write them something like this:

```
(adam) -[:MARRIED]-> (steve)
(steve) -[CHILD]-> (jaime)
(adam) -[CHILD]-> (jaime)
(jaime) -[:LIVES_AT]-> (2560 Ninth Street)
(jaime) -[:ORDERED]-> (order 2560)
(jaime) -[:ORDERED]-> (order 2561)
(jaime) -[:ORDERED]-> (order 2562)
```

Adam (well, the node representing Adam) has a married relationship to Steve (his person node). They have a kid named Jaime. She lives on Ninth Street and placed three different orders.

But You Promised Perl Code

For some reason I always feel compelled to give fair warning when I will be taking an approach that leads with a language that is not Perl, so here's your warning: Neo4j has a built-in graph query language called Cypher. Yes, I know, not a particularly encouraging name for a new thing to learn, but I didn't name it. It is possible to perform Neo4j actions from Perl without knowing any Cypher, but you won't get very far that way. This means we are going to dive into some basic Cypher first before getting to the Perl code. Sorry not sorry?

And being the meany I am, we already did it. The previous section had examples of nodes and relationships using Cypher conventions. Nodes are placed in parentheses. Relationships are specified in square brackets connected to nodes via arrows showing the direction of that relationship.

Quick aside about the directional nature of relationships: in Neo4j, you can only create relationships that go a single way from one node to another. But there is no restriction at query time around direction. You can easily (and with no performance hit) query either for relationships that exist (i.e., "Who is Steve married to?") and for nodes in a relationship that goes in a specific direction (i.e., "Who is Adam's kid?").

Let's see some more Cypher statements so you can get a sense of how data is inserted and queried in a Neo4j graph database. For these examples, we're going to use the example movie database that ships with the community (free) version of Neo4j (you can access it from the web console with ":play movies"). Adam, Steve, and Jaime are a lovely family, but let's play around with a larger data set.

First off, let's start by populating the database. Here are some of the lines from the script that creates a movie node and a number of people nodes:

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999,
tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne',
born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
```

To break this apart, let's pick on Keanu. We create a Keanu node with the label "Person". That node has two properties (his name and birthdate). Now let's add some relationships:

```
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (LillyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)
```

A little more dense, but if you can handle Perl data structures, surely a little punctuation won't throw you. Let's start from the bottom because those statements are simpler. The Wachowski sisters directed *The Matrix*, so we created DIRECTED relationships from them to their movie. Similarly, Joel Silver produced the movie, so there is a PRODUCED relationship put into place. Back to Keanu: Keanu acted in the film, so there is an ACTED_IN relationship specified. The part of that line which may look peculiar is this part:

```
{roles:['Neo']}
```

Earlier I mentioned that relationships are first-class citizens. They also have properties (just like nodes do). This is just attaching a property to that relationship. The property holds a list (actors can play more than one part in a movie), hence the square brackets in the property definition.

Just to make sure this is crystal clear (and because I find it so cool), not only are we specifying a relationship between a person node and a movie node (actor acted in movie), we also get to store information about that relationship (which roles or anything else we want) in that relationship definition itself. Relationships matter.

Once we load it into the database, we can ask the web interface to show us an interactive diagram of the database using the Cypher statement "MATCH (a) RETURN (a)" (match every node and returns them). By default the web interface only shows 300 nodes at a time, so the pretty picture in Figure 1 is only showing 300 nodes.

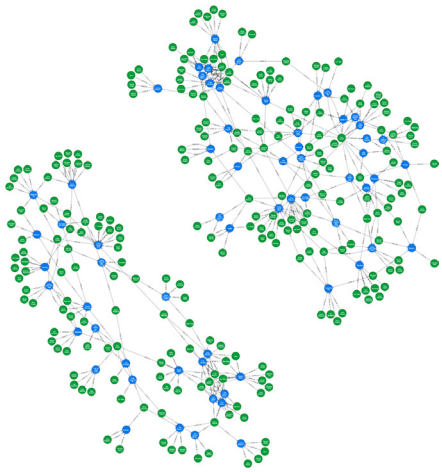


Figure 1: Three hundred of the nodes and relationships in our movie database

Now let's do some querying. We can find Keanu like this:

```
MATCH (k {name: "Keanu Reeves"}) RETURN (k)
```

Here we've said, "Find the node with that property, assign it to k, and return k." But this query doesn't really do the right thing. If there was a movie node (the blockbuster documentary on the life of Keanu), that would also get returned. Better would be to include the label in the query:

```
MATCH (k:Person name: "Keanu Reeves") RETURN (k)
```

If I run this from the command-line shell that ships with Neo4j (cypher-shell), I get the result I would expect:

```
Neo4j> MATCH (k:Person {name: "Keanu Reeves"}) RETURN (k);
+-------------------------------------------------------+
| k                                                     |
+-------------------------------------------------------+
| (:Person {born: 1964, name: "Keanu Reeves"}) |
+-------------------------------------------------------+
```

Now let's find his movies. To do this, we'll have to construct a query that includes a relationship:

```
MATCH (k:Person {name: "Keanu Reeves"})-[:ACTED_IN]->(kmovies)
    RETURN k.name,kmovies.title;
```

Breaking this down: find a Person node with the property "name" set to "Keanu Reeves," then look for all nodes related to that node by an ACTED_IN relationship. Return this, showing only the name property for the k nodes and the title property for the kmovies nodes. The result:

```
+-----------------------------------------------+
| k.name          | kmovies.title               |
+-----------------------------------------------+
| "Keanu Reeves" | "Something's Gotta Give" |
| "Keanu Reeves" | "Johnny Mnemonic"        |
| "Keanu Reeves" | "The Replacements"       |
| "Keanu Reeves" | "The Matrix Revolutions" |
| "Keanu Reeves" | "The Devil's Advocate"   |
| "Keanu Reeves" | "The Matrix Reloaded"    |
| "Keanu Reeves" | "The Matrix"             |
+-----------------------------------------------+
```

One last query, so let's make it a fun one. What if we want to find all of the actors Keanu acted with in the database? Let's break the question down first, then see it in Cypher form:

1. What movies has he been in? Just did that, check.
2. What actors have acted in those same movies? That can be stated as "Given a movie, what actors have an ACTED_IN relationship to that movie?"

Here's the Cypher version where our query asks both questions at once:

```
MATCH (k:Person {name:"Keanu Reeves"})-[:ACTED_IN]->
        (movie)
            <-[:ACTED_IN]-(actor) RETURN actor.name
```

I've broken it up on several lines to make it easier to read, but you would likely use it as a single line. We find the movie nodes representing the movies Keanu has acted in and then look for other actors who also have acted in each movie (have an ACTED_IN relationship to it). The results:

```
+--------------------------+
| actor.name             |
+--------------------------+
| "Diane Keaton"         |
| "Jack Nicholson"       |
| "Takeshi Kitano"       |
| "Dina Meyer"           |
| "Ice-T"                |
| "Brooke Langton"       |
| "Gene Hackman"         |
| "Orlando Jones"        |
| "Laurence Fishburne"   |
| "Hugo Weaving"         |
| "Carrie-Anne Moss"     |
| "Charlize Theron"      |
| "Al Pacino"            |
| "Laurence Fishburne"   |
| "Carrie-Anne Moss"     |
| "Hugo Weaving"         |
| "Emil Eifrem"          |
```

```
| "Laurence Fishburne"  |
| "Carrie-Anne Moss"    |
| "Hugo Weaving"        |
+-------------------------+
```

Chances are we only want to see each actor's name once, but I wanted to drive home the notion that we're walking around a graph to return results. To produce a list that has each actor listed only once, we would write:

```
RETURN DISTINCT actor.name;
```

There's a ton more things we can do using Cypher (for example, it becomes easy to do a "six degrees of Kevin Bacon" query), but I'd recommend you look at the Neo4j doc and demo packages for that information. Let's actually see some Perl.

DBI-ish...

We've explored DBI ("the standard database interface model for Perl") in the past, so hopefully this section elicits feelings of "Oh good, it is that easy." Before we actually look at REST::Neo4p, the Perl module we are going to use, I should mention that there does exist a DBD::Neo4p. This means you could use *exactly* the DBI syntax if you really wanted to. DBI is definitely the way to go when you are dealing with relational databases and you want to make sure that your code has some level of portability between database engines. I suppose it is plausible that you might be switching from a standard relational database, and so you will be treating the graph database like any other database engine initially, but this feels like a bit of a stretch for me. There might be another scenario I'm not thinking of, but, in the meantime, let's dive into REST::Neo4p. Even if it isn't DBI per se, it is definitely modeled on it.

Just as you would do with any DBI code, the first step is to connect to the database:

```
use REST::Neo4p;
REST::Neo4p->connect( 'http://127.0.0.1:7474',
                      'neo4j', 'password' );
```

From here we can go two ways with the module:

1. We could use method calls to operate on the database.
2. We could tee up and then execute Cypher commands in a very similar fashion to the way we might use SQL in a standard DBI program.

Let's see both approaches.

The first method would look like this:

```
my $movie = REST::Neo4p::Node->new(
    {
        title     => 'The Room',
        released  => '2003',
        tagline   => 'Experience this quirky new black comedy,
it's a riot!'
    }
)->set_labels('Movie');
my $person = REST::Neo4p::Node->new(
    {
        name    => 'Tommy Wiseau',
        born    => '1955',
    }
)->set_labels('Person');
$person->relate_to( $movie, 'DIRECTED');
```

We create nodes and their properties and relationships (with no properties—we would just need to add in a hash reference with the info, similar to the way it is done for node creation).

For the second approach, using Cypher from Perl, it should be relatively straightforward to people used to working with SQL from Perl:

```
my $cypher = REST::Neo4p::Query->new(
   'MATCH (theroom {title: "The Room"})<-[:DIRECTED]-(director)
       RETURN director');
$cypher->execute;
while (my $result = $cypher->fetch) {
       print $result->[0]->get_property('name'), "\n";
}
```

In this code we prepare and execute a Cypher query that

- starts by looking for the movie node with a property matching the title we are looking for,
- then looks for nodes that have a relationship of director to that movie and returns the nodes it finds.

The rest of the Perl code just iterates over the returned nodes and prints out the key property from them (the name of the director).

And that's the basics of working with Neo4j from Perl. Take care, and I'll see you next time.

Knowing Is Half the Battle

The Cobra Command Line Library of Go

CHRIS "MAC" MCENIRY

Chris "Mac" McEniry is a practicing sysadmin responsible for running a large e-commerce and gaming service. He's been working and developing in an operational capacity for 15 years. In his free time, he builds tools and thinks about efficiency. cmceniry@mit.edu

In our previous articles, we built a remote directory listing service. If we wanted to, we could extend this to provide generalized remote file system access by adding a `gcp`, `gmv`, `grm`, `gcat` or any other number of mirroring actions that you can do with a local file system and the command line. In this article, we're going to go through a little bit of an exercise to see what that would look like.

The straightforward extension of the `gls` command is to put the core of the actions into libraries and then copy the interface of those libraries into separate mains that become separate binaries. In working with Go, you may have also noticed that the binaries produced tend to be a little on the large side. A simple "hello world" built with Go 1.9 weighs in at around 2.5 MB. There's a lot in that binary, and it is a tradeoff between maintaining a separate runtime and dependencies versus having the runtime and dependencies come with the binary-space versus ease of distributing the application. While the size consideration is not an issue in most cases with current storage availability, there are various use cases where storage is still limited—embedded and RAM-based just to name a couple.

Several tools provide a way for you to have your cake (ease of distribution) and eat it too (limit overall size needs). We can combine all of the commands into a single binary, and then access it one of two ways. In tools like BusyBox, the same binary is invoked, and it looks at what it was called in order to decide what to do. Other tools, like Git, Vault, the AWS command line tools, and even Go itself, use subcommands to decide what to do.

In addition to deciding which action to take, the binary has to handle all of the arguments passed to it. In handling command line arguments, there are two categories: positional and optional. Positional arguments are ones whose meaning comes from where they show up on the command line. One could argue that subcommands are just a special case of the positional argument, and that the command name as subcommand is a special case on top of that. Optional arguments are ones that are identified by name and an argument token (the poster children have the token being a hyphen – or, in POSIX/LongOpt format, a double hyphen --). One of the added benefits of a combined binary is that you can more easily maintain consistency across your command line arguments, especially the ones that are needed across multiple binaries.

gofs

In this article, we're going to focus on building out the command-line interface to gls into a general tool called gofs. To make this a bit more concrete, we're going to use the following rules:

- It will be a single command line tool with subcommands (instead of examining the process name).
- It will have a general option, which will allow us to select different servers.
- It will implement the interface to `ls` and `cp` to show two patterns for handling positional arguments.
- It will combine the server start interface as well under the `serve` subcommand.

Note: For brevity, the focus here is strictly on the command line interface. Most operations of the actual gofs library and service would look the same as our existing gls-based ones, so I'll leave an actual implementation of the service as an exercise for the reader.

The code for this can be found at https://github.com/cmceniry/login/tree/master/gofs.

Go Command Line Options

Most programming decisions are opinions; the Go command line options are no different. There are several ways to achieve this. We're going to examine three options, and then choose one.

Standard Library: `os.Args`

As seen in the preceding articles, the most basic level is parsing the command line options directly. This involves handling the `os.Args` slice.

For the gls services, we just took the very first argument passed into our program and operated on that. There was no decision process, and it was very simple.

This has the upside of being very straightforward to process. The downside is that it is the application's responsibility to handle the various types of arguments. It has to

1. process and remove the optional arguments and then
2. decode subcommands/positional arguments.

Standard Library: flags

Go comes with an opinionated command line options-parsing library. It is strictly for parsing the command line arguments—i.e., it does not handle subcommands. It needs to handle the remaining arguments, directly or via another library, to be able to achieve the subcommand pattern that we're aiming for.

Probably the most controversial opinionated implementation of the flags library is what format it uses for arguments. It only handles the single hyphen token. Additionally, it does not support short options. Though you can name an option using a short name, there's no library method to connect a short and long option.

The Cobra Library

Steve Francia took the time to extract the subcommand pattern out into a library, and separately took the time to extend (more like a drop-in replacement for) the flags library to support the LongOpt format. These libraries are:

- https://github.com/spf13/cobra
- https://github.com/spf13/pflag

Cobra is billed as "a library providing a simple interface to create powerful modern CLI interfaces similar to git & go tools." pflag handles the the option parsing for Cobra built commands.

Given that Cobra handles the subcommand pattern that we want, and pflag implements the familiar LongOpt format, this seems like a good choice on top of which to build.

Implementation

Cobra works by defining Command structs and then wiring them together with flag arguments and with other commands. Both our primary command and all subcommands use the Command struct. It is the wiring that decides whether a command is a primary command or a subcommand.

To organize our code, we're going to implement our application as the `gofs` package. The primary, or root, command will be in its own root file. Each subcommand will also get its own file.

The `main.main` func will be in its own `cmd` directory. This is to reduce any confusion with a `main` package file being in the `gofs` package directory—some tools and IDEs will see this as an error.

Our directory structure looks like:

```
gofs/cmd/main.go # main.main calls into root.go
gofs/ls.go
gofs/mv.go
gofs/put.go
gofs/root.go     # primary command
gofs/serve.go
```

To build this, using the default `GOPATH`:

```
go get -u github.com/cmceniry/login/gofs
cd ~/go/src/github.com/cmceniry/login/gofs
go build -o gofs ./cmd/main.go
```

Root Command

Cobra builds commands off each other. At the start of it is the root command:

`root.go : rootcmd.`

```
var rootCmd = &cobra.Command{
```

Common convention is that each command tends to be a package-level variable. You could define these inside of a setup function (and we'll see something like that with the arguments), but the common method is to do this at the package level.

For each command, we first need to define its usage. We'll see additional usage options shortly, but to start, we need to describe what our application is:

root.go : rootusage.

```
    Long: `A simple interface to a remote file system.
It provides a remote interface similar to the standard tools of
ls, cp, mv, etc.`,
}
```

Cobra provides a built-in help system. `Long` is displayed whenever `help` or `--help` or invalid options are used.

```
$ go run cmd/main.go
A simple interface to a remote file system.
It provides a remote interface similar to the standard tools of
ls, cp, mv, etc.
```

Now that we have a root command, we need to wire it into the `main.main`. Since the root command is in the `gofs` package, we need to export that in some manner. The common way is to define an `Execute` func which `main.main` invokes:

cmd/main.go.

```
package main

import "github.com/cmceniry/login/gofs"

func main() {
   gofs.Execute()
}
```

The `gofs.Execute` func is really just a wrapper around `rootCmd.Execute`:

root.go : execute.

```
func Execute() {
   if err := rootCmd.Execute(); err != nil {
      fmt.Println(err)
      os.Exit(1)
   }
}
```

We cannot invoke `rootCmd.Execute` from `main.main` directly because `rootCmd` is a `gofs` package-level variable. While this may present extra hoops to jump through, it does keep the code cleanly separated. It's unlikely that any `main` using this will want to do anything else, but this does make it very clear that this command line library is meant to only operate in an opinionated way.

`--server` **Optional**

Since all of our subcommands are going to depend on being able to connect to the server, we have to put the configuration for that at the highest level. This will be an optional variable `--server`, which takes the Go network string (e.g., `localhost:4270`). This can be passed to every subcommand so that they are consistently connecting to the correct server.

Options are added to Cobra commands using the `*Flags()` methods. Cobra has `PersistentFlags`, which carry from commands to subcommands, and local `Flags`, which only apply to the subcommands. There are special rules for applying local `Flags` to parent commands and passing those along, but those aren't needed for this exercise.

To add an option, we first have to define a variable for where to keep the flag's value inside of our program. Like the commands, these are commonly found as package-level variables since they may be used in various parts of the package's code. Since it attaches at our `rootCmd`, we can include this in our `root.go` file:

root.go : serveraddress.

```
var (
   serverAddress string
)
```

We could choose to add sections to our `Execute` command to do some initialization, but it would be nice to have this happen a bit automatically. Conveniently, Go has a facility for this.

When Go instantiates a package inside of the runtime, it performs a few initializations. The first that we're concerned with is to set up package-level variables (e.g., `rootCmd`, `serverAddress`). Following that, Go invokes the package's `init` func. This order is critical to us because we need to be able to reference the package-level variables inside of our `init` func.

Our `init` func starts like this—we'll add more to it in a bit:

root.go : flag.

```
func init() {
   rootCmd.PersistentFlags().StringVarP(
      &serverAddress,
      "server",
      "s",
      "localhost:4270",
      "address:port of the server to connect to",
   )
```

Since we're binding only one option, we only need to invoke it once. PersistentFlags accesses the flags for the gofs root command. StringVarP binds a pointer for a string to a variable, so that the variable can be modified. We use the address of our serverAddress variable to act as our pointer. The next two arguments, "server" and "s", declare the name of the argument. Following that is the default value. The very last part is our usage information, which shows up in help much like the command.Long field.

The **serve** *Subcommand: No Arguments*

Now that we've built out our primary command, we can build out and attach a child to it.

The simplest of these is the command that would start up our gofs server: serve. We construct it very much like our rootCmd, although we're going to add more usage information to it:

server.go : servercmd.

```
var serveCmd = &cobra.Command{
   Use:   "serve",
   Short: "Start the gofs server side",
   Long: `This will run the gofs server.
The gofs server provides a remote file system management
interface.`,
```

There are two new fields here. Use describes the name of our subcommand. Short shows up when help is invoked on our rootCmd (as opposed to Long which shows up when help is invoked on this command).

Next, we declare that this command should have no position arguments.

server.go : args.

```
   Args: cobra.NoArgs,
```

While it's not clear here, cobra.NoArgs is a func, not a data value. You can supply your own argument validation function, or use the built-in ones that come with the Cobra library.

We can now provide the func for our command to run. As mentioned, since the focus of this article is on the command line interface, the command just returns output back to the user.

server.go : run.

```
   Run: func(cmd *cobra.Command, args []string) {
      fmt.Printf("Starting server on '%s'\n", serverAddress)
   },
```

We'll see shortly how the func arguments can be used.

The last part to do is to wire serveCmd to our rootCmd. Like the --server argument, this is done in the init func:

root.go : wireserve.

```
   rootCmd.AddCommand(serveCmd)
```

ls/mv**: Handling Arguments**

Next let's look at one argument with the ls command (one here to mirror the use of ls from previous articles). At this point, most of the ls command should be inferable:

ls.go : cmd.

```
var lsCmd = &cobra.Command{
   Use:   "ls [path to ls]",
   Short: "Shows the directory contents of a path",
   Long:  `Shows the directory contents of a path. If given
no path, uses the running path for the gofs server.`,
   Args:  cobra.MaximumNArgs(1),
   Run: func(cmd *cobra.Command, args []string) {
      fmt.Printf("ls server='%s' path='%s'\n", serverAddress,
args[0])
   },
}
```

The new item here is the Args value. cobra.MaximumNArgs is a built-in func that is used to indicate a cap to the number of positional arguments. Of special note to remember is that Args requires a func, so cobra.MaximumNArgs is a func that returns a func.

We can see a similar approach to Args with the mv command:

mv.go : cmd.

```
var mvCmd = &cobra.Command{
   Use:   "mv source ... target",
   Short: "Moves a file to a destination file, or moves
multiple files into a directory",
   Long: `If given two arguments, moves the first file to
the second file. If that second file is a directory, the first is
moved into it.
If more than two arguments are given, it moves all but the
last file into the last one which it expects to be a directory.`,
   Args: cobra.MinimumNArgs(2),
```

Note that Args is used to validate the arguments, not to decide how to use them. That happens in the Run field:

mv.go : run.

```
      Run: func(cmd *cobra.Command, args []string) {
         if len(args) == 2 {
            fmt.Printf("Moving file '%s' to '%s'", args[0],
args[1])
         } else {
            fmt.Printf(
               "Moving files '%s' into directory '%s'\n",
               strings.Join(args[0:len(args)-1], ","),
               args[len(args)-1],
            )
         }
      }
   }
```

In this case, we want to behave differently depending on the number of arguments. With two arguments, we're looking largely at a rename or move. With three or more arguments, we can only do a move. Since this is done at the same time as our normal execution, it happens all at once, unlike the separate validation step.

Before we finish, we have to wire these commands to our `rootCmd` in the package's `init` func:

root.go : wirelsmv.

```
      rootCmd.AddCommand(lsCmd)
      rootCmd.AddCommand(mvCmd)
```

Conclusion

This article has presented just the tip of the iceberg in handling command line arguments in Go. While there are several options, one of the most powerful ones is the github.com/spf13/cobra library. In addition to the work it performs for you, it also demonstrates some good aspects for the developer experience:

- Cobra allows you to keep all of your related code with itself. Each command is largely self-contained except where there are logical overlaps (PersistentFlags).
- It has a built-in help system that automatically generates help and usage results. The developer doesn't have to spend extra time managing a separate usage function, which means it's less work and less likely to go out of date.

I hope that you feel confident using the Cobra library in your own code. As mentioned, implementing the legwork of this code is left as an exercise for the user. Even if you don't implement the actual function, you can consider how other file system utilities might be implemented to work through some of the odd use cases of command line parameters.

Whatever path you choose, good luck and happy Going!

Save the Date!

usenix

LISA18

October 29–31, 2018
Nashville, TN, USA

LISA: Where systems engineering and operations professionals share real-world knowledge about designing, building, and maintaining the critical systems of our interconnected world.

The full program will be available in August.

Program Co-Chairs

Rikki Endsley
Opensource.com

Brendan Gregg
Netflix

www.usenix.org/lisa18

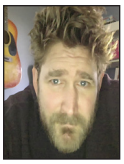
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

iVoyeur

Sensu Rising: An Interview with Matt Broberg

DAVE JOSEPHSEN

Matt Broberg is VP of Community for Sensu, Inc., focused on the incredible community around Sensu, the open source monitoring framework. Matt is on the board of the Influence Marketing Council, co-maintains the Evangelist Collective, contributes to the Go Community Outreach Working Group, occasionally blogs on Medium.com, and shares code on GitHub. He's also a fan of tattoos, rock climbing, and cats, though remains unsure of Schrödinger's.



Dave Josephsen is a book author, code developer, and monitoring expert who works for Sparkpost. His continuing mission: to help engineers worldwide close the feedback loop.
dave-usenix@skeptech.org

It's hard to believe that Sensu, the open-source, distributed monitoring framework, is over seven years old. Its scalable, ultra-flexible design and practitioner-focused development model still make it the most forward-thinking centralized poller in existence. The project is one of the very few that still "feels" fresh to me, and yet it retains that aura of bullet-proof resiliency that only comes with time in the trenches.

In recent months, the project founders have incorporated to form Sensu, Inc., hiring on a dream-team of people I personally adore, including engineers Greg Poirier (Opsee) and Jason Dixon of Graphite fame, as well as ace community architect extraordinaire, Matt Broberg. The fledgling corporation is busy funding and managing Sensu community development, providing enterprise Sensu support, and laying the groundwork for the future in the form of Sensu-Go, a ground-up rewrite of Sensu in the Go programming language.

Given all the exciting change happening in the Sensusphere, I thought it'd be fun to interview Matt and get a feel for what's going on from within.

Dave Josephsen: Describe Sensu in your own words.

Matt Broberg: Sensu provides total visibility for your business, from the server closet to the cloud. Said simply, Sensu connects the dots between every tool in your monitoring solution, providing a single way to manage service checks, telemetry, alerting, and remediation, and it gives you the right primitives to build custom monitoring that scales.

DJ: Who is using Sensu today? How many teams? How are they distributed with respect to industry?

MB: It's helpful to recall that Sensu has been around for seven years, while Sensu Enterprise has nearly three years (full story at [1]). With over 13,000 downloads a day of Sensu Core packages, we know there are more users than the team behind Sensu, Inc. has gotten to know, and we look forward to discovering them. Shameless request: if you're a current user, I'd love to hear from you: community@sensu.io.

Talks from last year's Sensu Summit are a great cross-section of our user base. We have companies large and small, off- and on-premises, running in every environment from bare metal to Kubernetes to AWS. You have folks like GoDaddy scaling a self-service Sensu environment with 40,000 clients spread throughout their globally distributed datacenters. Spin up a box, get base knowledge about your environment out-of-the-box, and then help your product team customize it all.

Then you have an architect at T-Mobile talking about Sensu monitoring their Cloud Foundry environment. Nagios migration is a very common use case for us. Sometimes it seems like everyone at the summit has a story about migrating from Nagios at some point due to scaling or customization challenges. My personal favorite comes from David Schroeder who goes into the detail of the how and why he needed to move on [2].

Like any healthy open core model, we have a majority of users successfully running on their own with the MIT-licensed open source version. A healthy majority of users are open source, using our large library of plugins filled with service checks and telemetry collectors, or running their pre-Sensu plugins for Nagios, or pushing data using the StatsD extension. Some significant OSS shops, like Yelp or TripAdvisor, have open sourced tools that make Sensu even easier to run. One of my favorites is Sens8, which extends Sensu functionality to fit smoothly into Kubernetes. Schuberg Philis [3], for example, has a great blog about the custom code they're running to monitor 20 Kubernetes clusters with Sensu and Sens8.

We have a growing number of Sensu Enterprise customers as well, who get the benefit of enterprise-y integrations (ServiceNow and Jira are popular) along with support and training. It's a perfect choice for those who want all the pieces of Sensu put together for them so they can focus on introducing monitoring to their teams. These companies run the gamut of company size, from those as large as GE, who use Sensu to monitor Predix infrastructure, to smaller organizations like David's I mentioned above. And what I personally love is that many of them are contributors to the community, answering questions for new users or by sharing plugins.

Sensu 1.0

DJ: Compared to other monitoring tools you generally compete with, what are Sensu's particular strengths?

MB: Sensu's strength is how well it meets the challenge of monitoring dynamic infrastructure. Whether you run on bare metal, hypervisors, container orchestrators, or clouds, no matter how short-lived or ephemeral, it all works with Sensu. Our client runs on all operating systems, everything from Windows and Linux to Mac OS, and we can ingest and emit monitoring data with any purpose-specific monitoring system out there.

Scalability is a big win for Sensu as well. Our clients participate in a pub/sub relationship with a scalable transport layer which, in turn, communicates with a scalable server layer. If you have more infrastructure to monitor, spinning up more Sensu servers linearly scales your processing of checks. Dynamic client registration has historically been a bit of a thorny problem in the monitoring world, but it's been a solved problem for Sensu from its inception. The Sensu client's dynamic self-registration (and deregistration) hits home for those who have felt the pain of getting alerts for servers that were deprovisioned long ago.

The last major category of users that I know will love Sensu are used to monitoring tools that only work when you use them in the exact way they were intended. Configuration of those tools becomes unmanageable as soon as you leave that happy path, and our users love customization. Sensu has sane defaults, but will also never give you that feeling of being limited. Its API exposes the right primitives to let you build the monitoring you need with all the event handling, filtering, and automation of bits you can imagine.

DJ: We understand Sensu was architected from the ground up with a scalable distributed architecture model. Can you give us a rough idea of what the architecture looks like? For example, what are the primary components of a Sensu install, and how do they work together to achieve visibility?

MB: The Sensu client is our heavy lifter; it collects measurements. Typically, it runs on the instance you want to monitor, but it can also interrogate remote entities like switches or act as a process-local endpoint to receive, for example, thread-level metrics from a locally running app. Clients can also cooperate with other clients to achieve summarization or route messages, commonly around things like firewalls.

Clients self-register with the Sensu transport layer, which, by default, is a RabbitMQ Queue. Clients use the transport layer to publish their check results to the Sensu server and consume new check requests from Sensu server or events from other clients.

The Sensu server orchestrates service checks by publishing check requests to, and collecting service check results from, various clients via the transport layer. Nagios style (OK,WARN,CRIT,UNKNOWN) checks, as well as metrics and telemetry collection, happen by the same means, via messages passed through the transport layer. The Sensu server stores its state in Redis, performing roughly a single write operation per check result. Every Sensu server in the installation uses the same Redis state DB, ensuring that each individual Sensu server is, itself, stateless.

Most users persist data beyond Sensu, which fits perfectly with the design. It was a design goal for Sensu to easily and cheaply route telemetry or check results to external time-series databases like Graphite, Librato, and InfluxDB or store output to logging platforms like ElasticSearch.

DJ: Sensu ships with a very nice RESTful API. What sorts of operations are available from the API, and how do your customers use it?

MB: One of Sensu's greatest strengths is its RESTful API, making all of the data captured by Sensu accessible via HTTP. This API-first approach is a huge win for those living in dashboards; users can query for everything from current events (i.e., incidents) to registered Sensu client information. The fact that everything—from the Uchiwa dashboard and CLI to third-party dashboards like Grafana—uses the same API to communicate provides a single authority to keep results consistent.

Beyond the dashboard, you have a ton of options when you think about the API. Customers are a quick curl loop away from silencing alerts or getting a snapshot of the client health. There are endless ways users can combine API calls to flesh out runbooks, then add links to them to your check results with a custom attribute. The world is your oyster.

DJ: Many monitoring systems are protective with the monitoring data they collect and inflexible with respect to exporting that data to other systems. Can you talk a little bit about Sensu's philosophy on interoperability?

MB: What personally attracted me to working at Sensu was the story of how we intentionally fit into a best-of-class set of monitoring software instead of trying to be everything to everyone. Sensu wants to help you build the monitoring pipeline you need. In most cases we can natively ingest check results from your existing plugins in foreign data formats (Nagios, StatsD, and now Prometheus through extensions) and output to an ever-growing litany of other monitoring systems (Graphite, OpenTSDB, Metrics 2.0, JSON, and more).

Through our pluggable architecture, you choose where your data lives: your favorite TSDB, SaaS, S3 buckets, or anywhere else. Same goes for on-call or escalation management through OpsGenie or VictorOps or otherwise. Sensu makes sure it gets there. You get to decide where it goes.

DJ: Can you give us a rough idea of how hard it is to migrate to Sensu from an existing monitoring system (like Nagios)?

MB: It's as easy as running an existing Nagios check in a Sensu check config file. If you have an existing plugin for Nagios, maybe through `yum install`, and if you want to run it in Sensu, you deploy Sensu—ideally through your favorite configuration management tooling—and then wrap the command into a check definition under the 'command' attribute. Ta-da, you're done.

Sensu 2.0

DJ: We understand Sensu2 is available via GitHub at https://github.com/sensu/sensu-go. What is the release date, and what is the current status of the release candidate?

MB: Sensu Core 2.0 is in an Alpha state as of today, making it the perfect time to dive in to make sure to get your feedback in to guide the user experience. I recommend spinning it up on a non-production environment and seeing how it goes. When we hit Beta, we'll have a fully documented API and some larger-scale test cases to point to for performance expectations. Our official release target is for later this year, but we are committed to production readiness being a gate to GA, not a date.

DJ: We understand Sensu 2 is a from-scratch rewrite in Golang. Can you talk about what prompted the rewrite and share any top-level goals you set out to accomplish?

MB: While extremely resilient and powerful, Sensu 1.x's dependencies are numerous. We knew an adjustment was necessary to get to where monitoring needs to go to manage bare metal alongside container and serverless workloads. Moving from the external dependence and runtime requirements to a simple two-binaries-and-you're-done design will have a major impact to ease of deployment.

Go, as a language, offers clear advantages for that future as well. Concurrency is straightforward with goroutines, and many features that are seen as advanced in other languages are baked into Go's suite of tools, like race detection, testing, and performance analysis to name a few of my favorites.

In recent years, Go has established itself as the new language of systems programming. Because of this, our users are increasingly learning Go as part of their development toward an SRE skill set, making it even more essential to ensure community participation. Go's popularity and growing community provide a wealth of shared knowledge and understanding. The language and its documentation are welcoming to this new, growing segment of users coming from higher-level interpreted languages and frameworks in Ruby and Python.

DJ: We were excited to hear that the datastore in Sensu2 will be changed from Redis to etcd. Has your experience with etcd been positive so far?

MB: Etcd has been a powerhouse of a datastore. Sensu backend will have etcd embedded for clustered state and configuration management, replacing the state that Redis managed and the configuration files that used to live on disk. It's exciting to get a highly available Sensu backend that has the thorough testing that etcd gets in order to support large-scale Kubernetes deployments. That's a slam dunk for us.

We've also really enjoyed our interactions with the team working on etcd. We've contributed a few patches and have seen turnaround on bug reports in just a few days. The last fix we worked on with them was in a release only two weeks after the bug was filed and fixed on master. It's wonderful to be part of that community as well.

DJ: To what extent, as a user of Sensu2, will I need to be an etcd adept? Is its presence completely abstracted away to the point where I don't even know it's there? Or will I be expected to perform light maintenance/tuning?

MB: Our goal with 2.x is to abstract project dependencies so users can focus on the goal of monitoring the right services. If we can stick to sane defaults and avoid exposing config we don't need, it will be for the better. Maybe that's idealistic and we'll need to allow users to tune etcd's configuration, but we'll cross that bridge when it's warranted. The core team is open to adapting as we continue to test in larger environments.

DJ: Given that the datastore is changing, is the transport layer, RabbitMQ, also changing or possibly going away entirely?

MB: Yes, it is. RabbitMQ is a great piece of technology, providing pub/sub messaging with queueing and several routing topologies. It also does way more than Sensu ever needed it to do. With Sensu 2.x, we've implemented a built-in messaging transport, greatly simplifying Sensu's architecture, while still having the key capabilities that RabbitMQ provided. Given that we only ever scratched the surface of the power of RabbitMQ, it made sense for us to simplify the architecture and build the little bit of queueing we need. Fans of simpler architectures will be happy to know we have the same pub/sub model without additional services to run.

DJ: What will the upgrade path between Sensu and Sensu2 look like? Can Sensu2 process events from an in-place Sensu client? Is Sensu2 plugin-compatible with Sensu? API-compatible?

MB: The team at Sensu, Inc. and the many community contributors are committed to making the migration path as simple as can be. That said, this major release will be a breaking change in a few ways. New Sensu clients, backend and dashboard, will be deployed during installation. They will no longer need RabbitMQ and Redis services alongside them, so these can be spun down as part of installation as well.

Most importantly, all your existing plugins will run on both versions of Sensu. We will have runtimes available for download so you can pick up an embedded Ruby environment to keep your plugins up-and-running. Client config and plugins will need to be deployed alongside the new binaries. Our main focus will be to release updated Ansible, Puppet, and Chef code to enable the majority of users to painlessly deploy Sensu Core 2.0. For others not living the infrastructure-as-code paradigm, we will have upgrade readiness guides and CLI tooling to ease the transition.

DJ: Where can I hang out with the Sensu community at large and/or perhaps contribute to the development of Sensu2?

MB: We would love to have you get involved. We have a #sensu2 channel to talk through user experience and give live feedback in our Community Slack (and yes, you can sign in using IRC!). For the day-to-day banter of software development, join the #core-dev channel. If—or should I say when—you have a great idea or a new bug to share with us, get involved on GitHub at https://github.com/sensu/sensu-go. Star the repo to let us know you're interested, or watch it to get regular notifications every step of the way. If you prefer the highlights over the details, sign up for our newsletter for regular updates.

References

[1] Caleb Hailey, "From Open Source to Open Core: The Hard Way," The Sensu Blog, May 2, 2016: https://blog.sensuapp.org/from-open-source-to-open-core-bc3007c96236.

[2] Sensu Summit 2017, YouTube: https://bit.ly/2GEcWsR.

[3] Andy Repton, "Our Journey Implementing Sensu to Monitor Kubernetes in Production," The Sensu Blog: https://blog.sensuapp.org/our-journey-implementing-sensu-to-monitor-kubernetes-in-production-5764aff2dd50.

For Good Measure
Remember the Recall

DAN GEER AND MIKE ROYTMAN

Dan Geer is the CISO for In-Q-Tel and a security researcher with a quantitative bent. He has a long history with the USENIX Association, including officer positions, program committees, etc. dan@geer.org

Michael Roytman is the Chief Data Scientist at Kenna Security. His work there focuses on cybersecurity data science and Bayesian algorithms. He serves on the board of the Society of Information Risk Analysts as Program Director. He is also a technical advisor in the humanitarian space, having worked with Doctors Without Borders, The World Health Organization, and the UN, and is a member of Forbes Technology Council. He is the cofounder and board chair of Dharma.ai, for which he landed on the 2017 Forbes 30 under 30 list. He holds an MS in Operations Research from Georgia Tech, and his coffee roastery, Sputnik Coffee Company, offers dog treats to all visiting canines. mikeroytman@gmail.com

> When you have eliminated the impossible, whatever remains, however improbable, must be the truth.—*Sir Arthur Conan Doyle*

Sherlock's statement is most often quoted to imply that uncommon scenarios can all be explained away by reason and logic. This is missing the point. The quote's power is in the elimination of the impossible before engaging in such reasoning. The present authors seek to expose a similar misapplication of methodology as it exists throughout information security and offer a framework by which to elevate the common Watson.

There was a time when one might answer the question, "What do you do?" with "computer security," but even five years ago generalists were beginning to be scarce, or, as one of us wrote:

> While some people like to say, "Specialization is for insects," tell me that the security field itself is not specializing. We have people who are expert in forensics on specific operating system localizations, expert in setting up intrusion response, expert in analyzing large sets of firewall rules using non-trivial set theory, expert in designing egress filters for universities that have no ingress filters, expert in steganographically watermarking binaries, and so forth. Generalists are becoming rare, and they are being replaced by specialists. This is biologic speciation in action, and the narrowing of ecologic niches. In rough numbers, there are somewhere close to 5,000 various technical certifications you can get in the computer field, and the number of them is growing thus proving the conjecture of specialization and speciation is not just for insects and it will not stop. [1]

A year ago, Oppenheimer & Co CISO Henry Jiang offered a visual for the specialization state of the security world [2]; it has 86 specialties (and commenters asked for more). Yet among the many specialties that make up security, all recognize the inherent uncertainty created by a sentient opponent, and all are currently grappling with one of two formulations of the same problem: (1) there is too much noise and not enough signal, or (2) there is a shortage of qualified security professionals, a shortage arguably made more acute by specialization, however logical specialization is in the face of security pressure as it now is.

Specialization has not proved to be a panacea; we are still beset by errors. Gorovitz and MacIntyre [3] explored a similar phenomenon in medicine by categorizing errors doctors were making into three types: failures of ignorance, failures of ineptitude (failing to apply knowledge that already exists), and necessary fallibility, a kind of prehistoric black swan that we shall not concern ourselves with here.

In our pursuit of knowledge, we have generated ineptitude. This is not an asseveration about the security industry; it is rather an allusion to a law, an analytic relationship between precision and recall in search problems [4, 5], and it is our claim that "search problems" is what information security is all about.

Explanatory aside:

> Library scientists use *precision* to mean what fraction of search results are actually useful and *recall* to mean what fraction of potentially useful results that the search actually returned. Epidemiologists know precision as *predictive value positive*, what fraction of positive tests actually have disease, and recall as *sensitivity*, what fraction of those with disease will test positive. As may be obvious, making recall (sensitivity) rise makes false positives rise, too. Conversely, as false positives rise, the precision (predictive value of a positive result) falls.

Security tools mostly deal with answering some form of the question, "Does this matter?" In vulnerability management, that question is, "Does this vulnerability pose a risk?" In incident response, that question is, "Was this a malicious event or a false positive?" In threat intelligence, it can be said as, "Is this indicator malicious or not?"

Each of these questions by whatever name is a statistical test, a classifier which vendors have tried to answer since before the time of KDD '99—a contest for "a predictive model capable of distinguishing between legitimate and illegitimate connections in a computer network" [6].

Accuracy, in the technical sense, has little meaning when searching for rare events. If one in a hundred machines is infected, I am 99% accurate when I routinely guess that "none of our machines are infected." Hence, we turn to measures of recall and precision when evaluating how "good" we are as an industry at answering these questions. The hope for all analytic tools is to reduce the number of errors the user makes. Are present-day security tools reducing failures of ignorance and failures of ineptitude sufficiently? We illustrate that the answer is a resounding "no." But there is hope. We also show that by measuring existing models through the lens of precision and recall, small changes to the models can have outsized impact on error reduction.

The perhaps foreseeable consequence of Ian Grigg's 2008 article "The Market for Silver Bullets" [7] has been a maddening proliferation of vendors, paralleling the proliferation of cybersecurity practice specialties. In 2016 alone, venture capital firms laid out $3.1 billion in funding 279 new security vendors [8]. We've "enjoyed" a 25% compound annual growth rate in venture money for cybersecurity over the past 13 years, with over $800 million placed in 2017 Q4 alone, but when a typical organization uses 50+ vendor products at once, the output of that instrumentation means an overload in the volume, velocity, and variability of the data that describes the ground truth we seek to classify. It is no surprise that most alerts are never examined. In the course of developing security tools, defenses, and processes, we as an industry have made one simple miscalculation—we have attempted to output truth, aka results, but instead have output a vast amount of noise and have overloaded our most precious resource—the security professional's time.

Generally speaking, security instrumentation seeks to improve models on its own grounds—by reducing false positives or by increasing the search space. While either approach is logical, it means that vendors always start their analyses at the theoretical level, move to the population level, and progress steadily towards the customer's environment. The result is overwhelming, and every additional product installed adds to the problem. It would be better to optimize for the capacity an organization has at hand and construct models with the feasible in mind. So far as the present authors know, every CISO survey ever taken has essentially found cybersecurity to be a lemons market, one where the buyer can't readily tell a low-quality product from a high-quality one. Investopedia then reminds us that "Ironically, [a lemons market] creates a disadvantage for the seller of a premium [product], since the potential buyer's asymmetric information, and the resulting fear of getting stuck with a lemon, means that he is not willing to offer a premium price even though the [product] is of superior value" [9].

Eliminating the Impossible

We offer a solution widely deployed in practice in other fields and talked about in the parlor rooms of security. If security tooling were to focus on analyst enablement, the approach to testing might be altered to resemble something more akin to medical practice—cost-effective multi-stage testing and process termination (see [10] for our prior work on testing)—multi-stage so as to be able to optimize test performance without incurring side-effects, and process termination when no therapeutic difference would follow from sharper diagnostic detail even if that detail were available for free.

Rare does not mean malicious, and building models specifically for very low base rate maliciousness means there is very little chance your positive test results are true positives. If the base rate of a non-malicious event, vulnerability, or indicator is high, we can be fairly certain that our test will categorize a non-malicious event as such—whatever remains after that is a new search space, where the base rate of "malicious" to "benign" is more evenly balanced and hence lends itself better to a second test, one where precision is the goal. In short, the first tests you must apply are not the ones that identify the malicious but the ones that identify the benign. It is conceivable that in a specialist-heavy field, recall is always > precision, perhaps because specialization increases hourly rates.

A Worked Example

Consider ACME, Inc., a fictitious organization constructed by sampling the data set we have at hand: Kenna Security's vulnerability scan data [11]. It contains 8,551,837 assets, 293 organizations, a median vulnerability count per asset of 116, a median monthly close rate of 25 vulnerabilities per asset, and a median monthly open rate of 20 vulnerabilities per asset (hence a net reduction of five vulnerabilities per asset per month).

The fictitious ACME has 10,000 assets; they vary from load balancers to Linux boxes to printers, and so forth. ACME has 1,160,000 vulnerabilities (116 x 10,000). ACME can remediate 250,000 vulnerabilities in a month (25 x 10,000) during which time another 200,000 vulnerabilities will be released (20 x 10,000), i.e., ACME has the capacity to reduce total vulnerabilities by 50,000 per month (5 x 10,000). ACME's goal is to remove the riskiest vulnerabilities from the organization's environment, so they turn to the 28% of the in-the-wild detected vulnerabilities that are ranked "critical" by CVSS, meaning 324,800 (1,160,000 x 0.28) vulnerabilities are ACME's first concern.

But 87.8% of those CVSS criticals are false positives (prior work at [12]), so ACME's meaningful effort towards security is limited to 39,626 (324,000 x (1 – 0.878)) vulnerabilities that are CVSS true positives. This number (39,626) is well within ACME's remediation capacity (39,626 < 50,000), but only if the remediation is somehow aimed only at true-positive vulnerabilities. However, if the level of effort required to remediate 50,000 vulnerabilities is spread across all those 324,800 vulnerabilities marked as critical, 85% (1 – 50,000/324,800) of ACME's investment will yield no useful result. Not only that, there will still be 33,525 ((324,800 – 50,000) x (1 – 0.878)) unremediated true positive vulnerabilities extant. They go on next year's budget…

Turning to the threat and incident use cases (and using the base rates in BalaBit's contextual security intelligence report [18]), ACME would collect about 6.78 billion raw log entries per month, process about 34% of those, getting it down to 2.26 billion processed log entries, and receive 17,300 alerts per month—one alert for each 130,635 processed log entries. ACME has the capacity to investigate 34% of those, that is to say 5,900 (17,300 x 0.34) alerts. They incur a false-positive rate of 18% while taking an average of seven minutes to decide whether an alert is or is not malicious, 688 (5,900 x 7/60) hours of work of which 124 (688 x 0.18) hours is wasted. ACME will correctly classify and investigate 4,800 (5,900 x (1 – 0.18)) of the 17,300 events a month, neglecting 9,363 ((17,300 – 5,900) x (1 – 0.18)) actually malicious events because that would require an additional 1,330 ((17,300 – 5,900) x 7/60) hours of labor to get to. If the organization could handle all 17,300 alerts, 363 (17,300 x 0.18 x 7/60) hours of their labor would be spent on false positives.

Capacity Optimality

The following should be treated as axioms:

1. We are data rich and signal poor.
2. Multi-stage testing cost-effectively increases both precision and recall.
3. Analyst time is the capacity constraint for most security problems (and Cybersecurity Ventures predicts 3.5 million unfilled cybersecurity positions in 2021).

When we say "Remember the Recall" in the alert scenario, "recall" means the percentage of alerts investigated that are in fact malicious. In the vulnerability problem, "recall" means the percentage of vulnerabilities that are identified as worth actually fixing. In both alerting and in vulnerability remission, false positives burn analyst time, our most precious resource. But suppressing false positives is not good enough to be "the" answer [13]. We have to go multi-stage.

The first-stage test has to find and dismiss absolutely the maximum number of benign markers be they alerts, vulnerability notifications, or whatever. This test has to be cheap, which is to say automated. It has to have no false negatives, that is, whatever it says is benign has to be benign. Epidemiologists call this "specificity." In our ACME example, one alert from 130,635 processed log entries illustrates strongly reduced search space—discarding the benign as fast as is possible, and there's a lot of benign to discard.

Where the first-stage test exists to throw out every datum it can so long as there are no false negatives, the second-stage test exists to select every datum it can so long as there are no false positives. The second stage is the analyst, the person with that seven-minute budget for selecting true, not false, positives. His or her tools can be much better if, and only if, the analyst plus tool combo is presented with a search space with the benign removed, that is the second stage can really be focused on recall (sensitivity). Medicine is riddled with this technique [14]. Legal document review [15] and payment fraud [16] are already there, too. And for those who want academic backup, see [17].

This framework is necessary to understanding the current state of security. We exist in a dual-stage testing regime. We are subject to a low prevalence (rare event) environment. To act rationally in this scenario, the first test must remove as many false negatives as it can. This necessarily implies automation in hopes of increasing the analyst's seven minutes to a more reasonable figure. To act with real foresight is to look to methods that automate the second test as well, saving analyst time for the highest quality, pre-cleaned data we can provide.

Assuming that our first test has, as we suggest, high specificity, it is then safe(r) to automate and bias the second test towards recall—meaning we work to solve failures of ignorance. But if we can automate the second test, we can then increase the amount of time the analyst can spend deciding—meaning we are working to solve failures of ineptitude. Perhaps then, and only then, will we get enough minutes back to spend those minutes chasing those rare birds, the black swans.

References

[1] D. Geer, "Trends in Cyber Security," Nov. 6, 2013: geer.tinho.net/geer.nro.6xi13.txt.

[2] Cybersecurity domains: www.linkedin.com/pulse/map-cybersecurity-domains-version-20-henry-jiang-ciso-cissp.

[3] S. Gorovitz and A. MacIntyre, "Toward a Theory of Medical Fallibility," *The Hastings Center Report*, vol. 5, no. 6 (December 1975), pp. 13–23: www.jstor.org/stable/3560992.

[4] L. Egghe, "The Measures Precision, Recall, Fallout and Miss in Function of the Number of Retrieved Documents and Their Mutual Interrelations," *Information Processing & Management*, vol. 44, no. 2 (2008), pp. 856–876: doclib.uhasselt.be/dspace/retrieve/22360/measures%202.pdf.

[5] S. A. Alvarez, "An Exact Analytical Relation among Recall, Precision, and Classification Accuracy in Information Retrieval," Boston College, Technical Report BCCS-02-01, 2002: pdfs.semanticscholar.org/d8ff/71a903a73880599fdd2c7be12de1f3730d29.pdf.

[6] "KDD Cup 1999: Computer Network Intruder Detection": www.kdd.org/kdd-cup/view/kdd-cup-1999.

[7] iang.org/papers/market_for_silver_bullets.html.

[8] Cybersecurity Ventures: cybersecurityventures.com/cybersecurity-market-report.

[9] "Lemons Problem," Investopedia: https://www.investopedia.com/terms/l/lemons-problem.asp.

[10] D. Geer, "Testing," *;login:*, vol. 39, no. 5 (October 2014): www.usenix.org/system/files/login/articles/login_1410_12_geer.pdf.

[11] Kenna Security, "How the Rise in Non-Targeted Attacks Has Widened the Remediation Gap," September 2015: www.kennasecurity.com/wp-content/uploads/Kenna-NonTargetedAttacksReport.pdf.

[12] D. Geer and M. Roytman, "Measuring vs. Modeling," *;login:*, vol. 38, no. 6 (December 2013): www.usenix.org/system/files/login/articles/14_geer-online_0.pdf.

[13] A. Mokarian, A. Faraahi, A. G. Delavar: "False Positives Reduction Techniques in Intrusion Detection Systems-A Review," *International Journal of Computer Science and Network Security*, vol. 13, no. 10 (October 2013): paper.ijcsns.org/07_book/201310/20131020.pdf.

[14] J. Liu, F. Chen, H. Yu, P. Zeng, L. Liu, "A Two-Stage Bayesian Method for Estimating Accuracy and Disease Prevalence for Two Dependent Dichotomous Screening Tests When the Status of Individuals Who Are Negative on Both Tests Is Unverified," *BMC Medical Research Methodology* (September 2014): bmcmedresmethodol.biomedcentral.com/articles/10.1186/1471-2288-14-110).

[15] http://www.nytimes.com/2011/03/05/science/05legal.html.

[16] J. Markoff, "Armies of Expensive Lawyers Replaced by Cheap Software," *New York Times,* March 4, 2011: https://pdfs.semanticscholar.org/2259/447c4ee67017999cbe0a539b86185e263eec.pdf.

[17] M. K. Buckland and F. C. Gey, "The Relationship between Recall and Precision," *Journal of the American Society for Information Science*, vol. 45, no. 1 (January 1994), pp. 12–19: www.researchgate.net/publication/220433788_The_Relationship_between_Recall_and_Precision.

[18] https://pages.balabit.com/rs/balabititsecurity/images/BalaBit-eCSI-magazine-201503-10-13.pdf.

/dev/random
Machine Learning Disability

ROBERT G. FERRELL

Robert G. Ferrell is a fourth-generation Texan, literary techno-geek, and finalist for the 2011 Robert Benchley Society Humor Writing Award. rgferrell@gmail.com

Machine learning is a hot topic currently, and for good reason: machines are pretty stupid. Maybe not all machines, but the ones around my house certainly fall well into sub-genius territory. Take, for instance, my aluminum can crusher. It really only has one job: using human-powered compression, reduce an empty aluminum vessel that can hold 12 ounces into one with a capacity of perhaps 1/30th that volume. Gravity then induces that flattened carcass to drop through an appropriately-sized slot into a waiting receptacle. Elaborate, it is not.

Yet as much as 5% of the time, the resulting crumpled disk is sufficiently misshapen that it remains in the chamber, mocking me and gravity in concert. This is not an acceptable failure rate for a mechanism barely above the wheel in complexity. Oh, there are some who in their hateful ignorance will claim operator error, but I snap my fingers in their puffy faces. All I do is pull a handle—what's to err? It's just mechanical spite, if you ask me.

Because I'm, like, really old now, I have vivid memories of one of the more classic stupid machines of yesteryear: the eight-track tape player. Heck, let's lump the eight-track tape itself in there too while we're at it. Basically, an eight-track tape (and this goes for some early computer tapes, as far as I care) is a huge Gordian knot that sort of spirals in around itself on a spool mounted in a rectangular cartridge. It is designed specifically to be nearly impossible to open without incurring damage and fifteen minutes in the penalty box.

The innermost edge is where the tape pops out of the spiral and scrapes its way back across the top of the coil to some rollers that hold it in position for the magnetic read heads in the player. The whole thing looks like it was designed to work exactly once, which isn't too far off the mark. I know this mechanism intimately because, being a foolhardy manner of fellow, I made a little money in high school unjamming these things for my music-loving friends. It was not a pastime for the faint of heart or unsteady of hand.

The tapes themselves were already prone to spontaneous breakage, but just to make certain any music that managed to make it through was not enjoyed without significant additional risk, the industry provided each user with a machine winched up from the worst neighborhood of the Hadean realm. The eight-track "player" fed the staggeringly bad engineering of the tape cartridge through the guts of a repurposed pasta machine in the decidedly optimistic hope that whatever music might be encoded on the tape somehow leaked out into the amplifier before the self-destruct sequence was complete.

I offer up the eight-track as a machine learning bullet point mostly because it effectively taught anyone who came into contact with said machine to avoid it. While a couple of frustrating hours spent fishing mangled tape from around a variety of rollers accessible only via a narrow slot guarded by a spring-loaded door could mar even the sunniest of dispositions, the lessons thus imparted stuck around for life. Relying on such a system for one's music is hardly my idea of La Vida Bella.

There are in fact multiple areas of domestic life where I think machine learning might prove beneficial. My home security system leaps to mind. For some months I was plagued with an alarm that would go off for no readily apparent reason, almost always in the dead of night. A technician would be dispatched, certify that absolutely nothing was amiss with the wiring or electronics, and almost before he turned the street corner after departing the blaring siren would sound again.

When conventional technical solutions failed to present themselves, I began to suspect something supernatural was at work here. After all, if I'm going to be faced with an intractable technical issue I may as well be entertained by consideration of its origin, am I right? Of course I am. The problem with the otherworldly causation hypothesis is that this is a fairly new house and hasn't had time to build up any spectral inventory to speak of. Not much has passed to the other side here, apart from one cat and a host of houseplants my wife either inundated mercilessly or neglected to water at all.

Proceeding on the assumption that cats in the afterlife have no more initiative than they did while earthbound, my alarm system is not possessed by a feline spirit. Our recently departed Fenchurch simply wouldn't bother. Oh, she might bat some ghostly creepy-crawly in front of a motion sensor from time to time between celestial naps, but I can't see her intentionally messing with the Master Bedroom Door contacts on a regular basis unless their ectoplasmic manifestation happens to resemble feathers dangling from strings.

My alarm system could therefore probably benefit from some machine learning. But are the eponymous machines undertaking said learning, or meting it out? I'm not clear on this point. I suppose either situation would be an improvement. Sooner or later the machines are going to cut humanity completely out of the educational cycle. We'll have virtual machines going door-to-door (or node-to-node) offering tutorial services for a reasonable fee payable in whatever currency machines find pleasant to exchange (CPU cycles, maybe, or little bags of qubits).

There's been a lot of fairly hysterical speculation over the past few decades on when the digital singularity will take place and the dire consequences of said event for the human race. Some say the machines will assume control and immediately turn on us meat sacks, regarding us as no better than ants invading the picnic of existence. Others suggest that our machine overlords will be benevolent, granting humanity a largely unfettered life so long as we don't overstep our bounds and interfere with their ascendancy.

Even if I live to see that day, I'll be too old to care one way or the other. I'll just sit on the porch waving my gnarled fist defiantly and yelling at those damn bots to stay off my data.

Book Reviews

MARK LAMOURINE

Sams Teach Yourself Go in 24 Hours

George Ornbo
Pearson Education, 2018, 464 pages
ISBN 978-0-672-33803-8

You might think that *Sams Teach Yourself Go in 24 Hours* expects you to binge learn a programming language overnight. What Ornbo has in mind is much more reasonable and actually quite digestible. The whole line of Sams' Teach Yourself series runs to over 80 titles, so they have some experience with this format, and the polish shows. These are definitely practical learning manuals for professionals.

Ornbo's book is a prime example of the form. There is very little space given for theory or abstract concepts. The longest chapter is 22 pages, with most around 15 pages. All of the source code shown in the book is also available on GitHub (divided by chapter). Nearly every page has a small box labeled "Try it yourself." These boxes explain the examples, provide instructions, and describe the expected results for the reader.

Each chapter concludes with a short Q&A, a "Workshop" section with quiz questions meant to get the reader to think about or review the chapter, and finally three or four suggested exercises. Some of the exercises are coding practice, but others refer the reader to a video (e.g., Rob Pike's introduction to Go [1]) or further reading (e.g., blog posts, module documentation).

The author really means for you to read each chapter in an hour, spend some time working on the examples, and stop. I'm surprised to find I actually like this format. I'm used to skimming and scanning tech books in a sitting or two and only going back when I think, "Oh, I saw how to do that somewhere." By limiting each chapter to a small topic with tight examples, Ornbo creates truly bite-sized pieces. The upfront time limit reminded me to stop and play and digest each bit. I'm familiar with Go, so I didn't stick to it for all three weeks, but I think that a disciplined reader would do well to take it slow and enjoy the easy pace. I found myself thinking through new possibilities and experiments at the end of each session. The "workshop" and reading exercises do a good job of inspiring curiosity and provide the means to follow up.

Another advantage to the compact "24 hours" format is that it gives Ornbo the chance to talk about more than just the language syntax. He finished the pure language instruction halfway through the book, with chapters on goroutines and channels as Chapters 12 and 13. In the latter part of the book, he spends a chapter on debugging, one on packaging, two on building web services, and one on JSON serialization. There are even two "bonus chapters," available only in the eBook forms, that describe designing a RESTful API and a chat server using the concepts and techniques presented earlier.

Teach Yourself Go won't replace Donovan and Kernigan's *The Go Programming Language* [2], but it makes a great companion for the professional adding a tool to their box.

Crucial Conversations, 2nd ed.

Kerry Patterson, Joseph Grenny, Ron McMillan, and Al Switzler
McGraw-Hill, 2012, 244 pages
ISBN 978-0-07-177530-4

When I saw the cover of *Crucial Conversations,* where it proclaims "3 Million Copies Sold," my expectations were low. When your number-one recommendation is "We Sold a Lot," I'm prepared to be unimpressed. The whole cover screams "Pop Psychology Self-Help!" All of my skeptic buttons were pushed.

Crucial Conversations is indeed a self-help book but perhaps one that many people in tech could use. The major message is "Think before you speak, especially in high-stakes situations," and I know that's something I'm still learning.

In each chapter, the authors address some aspect of how people interact, in small groups or one on one. First, they introduce a scenario and the participants. Each person comes to the conversation with something at stake, but all have different goals, priorities, and points of view.

At the start of each conversation it is clear that things could easily go off the rails without some care. One person is the primary actor, usually the one with the least power and the most to lose. Through the remainder of the chapter the authors talk about the motivations of each person and how they might react to different approaches to the conversation, with a focus on the goals that the primary actor has and the pitfalls to avoid.

The chapters address how to approach a sensitive conversation and keep the goal in mind, while avoiding the minefields of emotion, defensiveness, cheap jabs, and destructive criticism. The goal isn't to win an argument but to have a dialog and come to a consensus.

I've been reading and learning a lot in the last decade about how my perspective and behavior, both in the workplace and at home, influence the impressions and responses of the people around me. Looking back on my career, I wish I'd understood some of these things long ago, and I struggle to act only in a way that will be constructive (something I did not always do).

The outward trappings of *Crucial Conversations* are those of an upscale business-oriented miracle cure for the climbing executive. The writing style has a similar feel throughout, right down

to the subsection at the end of the first chapter entitled "Our Audacious Claim," in which the authors talk about the ways you can "Improve Your Relationships" and "Improve Your Health" using their methods.

Crucial Conversations is one of several books by these authors. Together they have formed a company that makes its money selling the books and teaching seminars all over the country. The book feels a lot like a promotion for the seminar, and I came to read it after a friend suggested to her whole company that they should invite the authors to present. I'll find out more after that occurs.

There is, however, a contrast in content to typical diet, exercise, and pop-psych fads. The scenarios presented here are realistic and not too contrived. The authors really make few magic promises in the rest of the book, concentrating more on how each conversation requires listening, analysis, and self-control to reach the best outcome. Much of the messaging is in line with the tenets and goals of modern Agile management styles.

I don't know if there is a better way to present these ideas, but if readers can set aside their own wariness, they may find something here to think about and use. I certainly would recommend this book to someone who wants to think more about their own approach to communicating in tense situations.

Linux Hardening in Hostile Networks

Kyle Rankin
Pearson Education, 2018, 242 pages
ISBN 978-0-13-417326-9

One of the ideas I learned in the 1990s working as a sysadmin for a large ISP is "always assume your environment is hostile." It seems a bit quaint now, but we ran exposed server hosts and desktops and had to learn how to probe and harden them. We assumed our networks were unsafe, and our network group assumed the same thing about all of our infrastructure hosts. We weren't actively malicious to each other, but we were always wary of our assumptions. It actually made for a great dynamic, and we only had one incident I can recall where someone failed to pay attention and was caught out by the other group.

These days most people rely on their firewalls, both software and hardware, and on virus scanners. I'm not as actively paranoid as I once was just because I don't have the time. But I haven't stopped being watchful and curious.

When I picked up *Linux Hardening* I was surprised at how slim it is. I was expecting, for example, a series of rather arcane steps to tweak the behavior of the network stack in the kernel. Apparently, that kind of manual is why Rankin decided to write his own book. In his treatment he does touch on all of the expected topics, but he focuses on the most basic and practical steps to take in each area. What Rankin has realized is that, in the common focus on network boundaries and kernel exploits, a lot has been lost. It's not easy to find a high-quality guide to good, basic technical hygiene.

In the first major section of the book, Rankin offers the topics you'd expect: workstation and server security, host networking, and firewalls. The next four chapters touch on common services; web, email, DNS, and database. He closes with a chapter on incident response and a couple of appendices that go into some technical details of Tor and TLS/SSL.

Each chapter presents a progression from the most basic considerations through intermediate tasks and concludes with what Rankin believes are steps that can protect against even government-level attacks (with the caveat that today's advanced attacks are tomorrow's script-kiddie tools).

Rankin includes a number of techniques that make perfect sense today but have only become reasonable in the last decade or so. Tails is a modern bootable read-only distribution, and Qubes is designed to isolate each application in its own virtual machine. Both allow the user to control when data is written to disk or is shared between applications. In the networking section, Rankin shows how to create a personal VPN using OpenVPN and introduces Tor. The fact that both have become reasonable things for an intermediate level sysadmin to do just warms my geeky sysadmin heart.

Other high points are the inclusion of a treatment of email transport security using SPF, DKIM, and DMARC, and an entire chapter on securing DNS. The DNS chapter starts with preventing your servers from being co-opted into DDoS attacks and ends with the most practical, compact introduction to DNSSEC that I've ever read.

Rankin chooses several tools in the workstation section that have many peers. I would have liked to see some space given to some of the alternatives to his selections. That said, I understand the desire to avoid confusing readers with variety that perhaps doesn't really improve the learning experience.

Linux Hardening isn't a deep or comprehensive guide to Linux security, and I wish it had more references to external materials, but it is a good, broad survey of the most important topics. The basic and intermediate tasks are probably within reach of junior or intermediate sysadmins, and the advanced ones would make a good challenge. For the advanced sysadmin there are probably a few nuggets here. I'm off to create a VPN for my home network.

References

[1] https://www.youtube.com/watch?v=rKnDgT73v8s.

[2] *The Go Programming Language* (Addison-Wesley Professional, 2015), ISBN 978-0-134-19044-0.

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

Free subscription to *;login:*, the Association's quarterly magazine, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, networks and operating systems, and book reviews

Access to *;login:* online from December 1997 to the current issue: www.usenix.org/publications/login/

Discounts on registration fees for all USENIX conferences

Special discounts on a variety of products, books, software, and periodicals: www.usenix.org/member-services/discount-instructions

The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers

For more information regarding membership or benefits, please see www.usenix.org/membership/or contact office@usenix.org. Phone: 510-528-8649.

USENIX Board of Directors

Communicate directly with the USENIX Board of Directors by writing to board@usenix.org.

PRESIDENT
Carolyn Rowland, *National Institute of Standards and Technology*
carolyn@usenix.org

VICE PRESIDENT
Hakim Weatherspoon, *Cornell University*
hakim@usenix.org

SECRETARY
Michael Bailey, *University of Illinois at Urbana-Champaign*
bailey@usenix.org

TREASURER
Kurt Opsahl, *Electronic Frontier Foundation*
kurt@usenix.org

DIRECTORS
Cat Allman, *Google*
cat@usenix.org

David N. Blank-Edelman, *Apcera*
dnb@usenix.org

Angela Demke Brown, *University of Toronto*
demke@usenix.org

Daniel V. Klein, *Google*
dan.klein@usenix.org

EXECUTIVE DIRECTOR
Casey Henderson
casey@usenix.org

Results of the Election for the USENIX Board of Directors, 2018–2020

The newly elected Board will assume office on July 1, 2018.

PRESIDENT
Carolyn Rowland, *National Institute of Standards and Technology (NIST)*

VICE PRESIDENT
Hakim Weatherspoon, *Cornell University*

SECRETARY
Michael Bailey, *University of Illinois at Urbana-Champaign*

TREASURER
Kurt Opsahl, *Electronic Frontier Foundation*

DIRECTORS
Cat Allman, *Google*

Kurt Andersen, *LinkedIn*

Angela Demke Brown, *University of Toronto*

Amy Rich, *Nuna, Inc.*

Notice of Annual Meeting

The USENIX Association's Annual Meeting with the membership and the Board of Directors will be held at 6:00 pm on Tuesday, July 10, in Boston, MA, during the 2018 USENIX Annual Technical Conference.

First Impressions on the Path to Community Engagement

Liz Markel, Community Engagement Manager

On Day 7 of my new job at USENIX this past March, I found myself wandering the hallways of the Hyatt Regency Santa Clara taking in the energy and excitement of SREcon18 Americas. I have planned and attended many large conferences before—some with upwards of 20,000 people—but always with program content that was already near and dear to my heart, including nonprofit leadership, literature, and librarianship. The experience of being a non-tech expert at a tech-focused conference was completely foreign to me, though I was determined to leverage my fresh perspective to my advantage.

SREs are one of the many communities USENIX serves, and that I will too in my new role as Community Engagement Manager. As the new kid in town—with a background in marketing and nonprofit management, and only slightly more knowledge of tech than the average person—I wondered how I would find common ground with this exceptionally smart, passionate group of people, as well as the many other communities represented at conferences like FAST, NSDI, USENIX Security and others.

As I listened in on workshops, presentations, "hallway track" conversations, and the lively exchanges at vendor booths, I did indeed discover that we have much in common! We are both, to quote USENIX Board of Directors member Cat Allman writing in the Winter 2016 issue of *;login:*, "honest, practical, problem solving-engineers." The things I have built throughout my career differ significantly from what you, reader, are building with your daily work. However, I think these shared attributes are a fantastic foundation for good conversation.

SREcon18 Americas attendees built relationships and made connections in the Recharge Lounge sponsored by Twitter.

Conversations about what, you ask? I'd like to talk about what we at USENIX can build for you.

Do you have ideas to help fuel our community engagement activities? What do you want to see brought to life that will enhance your experience as a part of the larger USENIX community? A few opportunities come to mind that build on existing frameworks:

- Enhance engagement activities at conferences through like-minded groups such as Birds-of-a-Feather sessions (BoFs);
- Take advantage of effective tools for thoughtful discussion and networking online between conferences;
- Create content for USENIX's digital platforms or print publications that piques your interest and could spark lively conversation with your global colleagues.

This is a fascinating time to be part of an association staff as a team member focused on community engagement. The broadly observed trend for many professional associations shows interest in official membership waning among upcoming generations. This trend, coupled with natural shifts due to members' retirements, has challenging implications for member recruitment and retention. My initial observations lead me to believe that USENIX may be defying this trend, and that the root cause of this success is the thoughtful, pragmatic, and considerate organizational leadership that includes both the Board of Directors and the USENIX staff. This effective leadership manifests itself in many ways including the Conference Code of Conduct; the well thought-out rationale for making changes to the LISA conference for 2018 in response to attendee feedback and changes in the industry; and the personal, daily experiences

Tammy Butow of Gremlin kicked off "Chaos Engineering Bootcamp" at SREcon18 Americas with some fun audience participation.

Attendees at "Chaos Engineering Bootcamp" took sides in a brief debate about the value of chaos engineering for a business.

Many SREcon18 Americas sessions involved peer-to-peer collaboration and problem-solving.

Lisa Carey of Google delivered important skills in her workshop, "Tech Writing 101 for SREs."

Collaboration extended beyond the programs at SREcon18 Americas--teams could often be found problem-solving in other areas of the hotel.

Kate Taggart of HashiCorp promoted the benefits of newbies in her SREcon18 Americas program "Junior Engineers Are Features, Not Bugs."

Thomas Limoncelli delivered a humorous closing keynote for SREcon18 Americas: "Operational Excellence in April Fools' Pranks."

of USENIX staff—myself included. All-encompassing leadership of this caliber is extremely rare in the nonprofit sector, and it bodes extremely well for the organization's sustainability and continued relevance to the communities it serves.

The highly relevant mission of USENIX is another primary reason for the association's strength. Through our work, we are committed to:

- Fostering technical excellence and innovation
- Supporting and disseminate research with a practical bias
- Providing a neutral forum for discussion of technical issues
- Encouraging computing outreach into the community at large.

I spent much of my time at SREcon engaging attendees in conversations about their professional priorities, their impressions of USENIX, and their thoughts about trends in the field, all in relation to our mission. There were lots of interesting insights, but I was most impressed by the passionate support for USENIX's commitment to open access and the role this plays in fostering technical innovation and education. It's a passion that I am looking forward to exploring, understanding, and enhancing in the months and years to come.

I reflected on the importance of this shared passion as I browsed the Spring 2016 issue of this magazine, in which Board of Directors member Dan Klein described the glue that binds USENIX together and has kept it relevant for more than 35 years: "Magic, wonder, and play."

Where have you felt the magic, wonder, and play at work in your USENIX experiences? How can we scale it, share it with others across the country and around the globe, and grow the breadth and depth of our communities?

I would love to hear from you. Drop me a line (liz@usenix.org) so that we can start the conversation and build some amazing new things together.

Following her presentation with Jez Humble at SREcon18 Americas, Nicole Forsgren signed copies of her new book *Accelerate: The Science of DevOps*.

SREcon18 Americas program co-chairs Kurt Andersen (LinkedIn) and Betsy Beyer (Google) offered an appreciative shout-out to their fellow program committee members.

Attendees enjoyed the beautiful northern California weather during evening receptions at SREcon18 Americas.

SAVE THE DATES!

SRECON® ASIA AUSTRALIA

JUNE 6–8, 2018 • SINGAPORE
www.usenix.org/srecon18asia

SRECON® EUROPE MIDDLE EAST AFRICA

AUGUST 29–31, 2018 • DUSSELDORF, GERMANY
www.usenix.org/srecon18europe

SRECON® AMERICAS

MARCH 25-27, 2019 • BROOKLYN, NY, USA
www.usenix.org/srecon19americas

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

Register Now!

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Fourteenth Symposium on Usable Privacy and Security

Co-located with USENIX Security '18
August 12–14, 2018 • Baltimore, MD, USA

The Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018) will bring together an interdisciplinary group of researchers and practitioners in human computer interaction, security, and privacy. The program will feature technical papers, including replication papers, workshops and tutorials, a poster session, and lightning talks.

Workshops include:

- **Workshop on Security Information Workers (WSIW)**
- **Workshop on the Human Aspects of Smarthome Security Privacy (WSSP)**
- **Who Are You?! Adventures in Authentication (WAY)**
- **Workshop on Inclusion Privacy and Security (WIPS)**
- **Designing Privacy and Security Tools for Children and Teenagers**

Register by July 23 and save!

www.usenix.org/soups2018

Save the Date!

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

OSDI '18

13th USENIX Symposium on Operating Systems Design and Implementation

October 8–10, 2018 • Carlsbad, CA, USA

OSDI brings together professionals from academic and industrial backgrounds in what has become a premier forum for discussing the design, implementation, and implications of systems software. The OSDI Symposium emphasizes innovative research as well as quantified or insightful experiences in systems design and implementation.

Program Co-Chairs:
Andrea Arpaci-Dusseau, *University of Wisconsin—Madison*
and Geoff Voelker, *University of California, San Diego*

The full program and registration will be available in August.

www.usenix.org/osdi18

NSDI '19: 16th USENIX Symposium on Networked Systems Design and Implementation

February 26–28, 2019 • Boston, MA, USA

Sponsored by USENIX, the Advanced Computing Systems Association

Important Dates

Spring deadline:

- Paper titles and abstracts due: **Thursday, May 24, 2018, 6:00 p.m. US PDT**
- Full paper submissions due: **Thursday, May 31, 2018, 6:00 p.m. US PDT**
- Notification to authors: **Saturday, July 28, 2018**
- Final paper files due: **Wednesday, October 17, 2018**

Fall deadline:

- Paper titles and abstracts due: **Thursday, September 13, 2018, 6:00 p.m. US PDT**
- Full paper submissions due: **Thursday, September 20, 2018, 6:00 p.m. US PDT**
- Notification to authors: **Monday, December 3, 2018**
- Final paper titles due: **Friday, February 8, 2019**
- Final paper files due: **Wednesday, February 13, 2019**

Conference Organizers

Program Co-Chairs

Jay Lorch, *Microsoft Research*
Minlan Yu, *Harvard University*

Program Committee

Fadel Adib, *Massachusetts Institute of Technology*
Aditya Akella, *University of Wisconsin—Madison*
Katerina Argyraki, *École Polytechnique Fédérale de Lausanne (EPFL)*
Aruna Balasubramanian, *Stony Brook University*
Sujata Banerjee, *VMware Research*
Kai Chen, *Hong Kong University of Science and Technology*
Mosharaf Chowdhury, *University of Michigan*
Anja Feldmann, *Technische Universität Berlin*
Bryan Ford, *École Polytechnique Fédérale de Lausanne (EPFL)*
Michael J. Freedman, *Princeton University*
Roxana Geambasu, *Columbia University*
Monia Ghobadi, *Microsoft Research*
Jana Giceva, *Imperial College London*
Ronghui Gu, *Columbia University*
Haryadi Gunawi, *University of Chicago*
Haitham Hassanieh, *University of Illinois at Urbana-Champaign*
Jon Howell, *Google*
Rebecca Isaacs, *Twitter*
Xin Jin, *Johns Hopkins University*
Srikanth Kandula, *Microsoft Research*
Manos Kapritsos, *University of Michigan*
Dejan Kostić, *KTH Royal Institute of Technology*
Ramakrishna Kotla, *Amazon Web Services*
Arvind Krishnamurthy, *University of Washington*
Harsha Madhyastha, *University of Michigan*
Dahlia Malkhi, *VMware Research*
Allison Mankin, *Salesforce*
Derek Murray, *Google*
Aurojit Panda, *New York University*
KyoungSoo Park, *KAIST*
Amar Phanishayee, *Microsoft Research*
Raluca Ada Popa, *University of California, Berkeley*
Lili Qiu, *University of Texas at Austin*
K. K. Ramakrishnan, *University of California, Riverside*
Michael Schapira, *Hebrew University of Jerusalem*
Cole Schlesinger, *Barefoot Networks*
Vyas Sekar, *Carnegie Mellon University*
Ankit Singla, *ETH Zurich*
Anirudh Sivaraman, *New York University*
Alex C. Snoeren, *University of California San Diego*
Michael Stumm, *University of Toronto*
Ryan Stutsman, *University of Utah*
Geoff Voelker, *University of California San Diego*
Hakim Weatherspoon, *Cornell University*
John Wilkes, *Google*
James Hongyi Zeng, *Facebook*
Irene Zhang, *Microsoft Research*
Xinyu Zhang, *University of California San Diego*
Heather Zheng, *University of Chicago*
Lin Zhong, *Rice University*

Steering Committee

Aditya Akella, *University of Wisconsin–Madison*
Katerina Argyraki, *EPFL*
Sujata Banerjee, VMware Research
Paul Barham, *Google*
Nick Feamster, *Princeton University*
Casey Henderson, *USENIX Association*
Jon Howell, *Google*
Arvind Krishnamurthy, *University of Washington*
Jeff Mogul, *Google*
Brian Noble, *University of Michigan*
Timothy Roscoe, *ETH Zurich*
Srinivasan Seshan, *Carnegie Mellon University*

Overview

NSDI focuses on the design principles, implementation, and practical evaluation of networked and distributed systems. Our goal is to bring together researchers from across the networking and systems community to foster a broad approach to addressing overlapping research challenges.

NSDI provides a high-quality, single-track forum for presenting results and discussing ideas that further the knowledge and understanding of the networked systems community as a whole, continue a significant research dialog, or push the architectural boundaries of network services.

Topics

We solicit papers describing original and previously unpublished research. Specific topics of interest include but are not limited to:

- Highly available and reliable networked systems
- Security and privacy of networked systems
- Distributed storage, caching, and query processing systems
- Energy-efficient computing in networked systems
- Cloud/multi-tenant systems
- Mobile and embedded/sensor applications and systems
- Wireless networked systems
- Network and workload measurement systems
- Self-organizing, autonomous, and federated networked systems
- Managing, debugging, and diagnosing problems in networked systems
- Virtualization and resource management for networked systems
- Systems aspects of networking hardware
- Experience with deployed networked systems
- Communication and computing over big data on networked systems
- Practical aspects of economics and verification applied to networked systems
- Any innovative solution for a significant problem involving networked systems

This year, we're making two major changes: we're offering two submission deadlines and we're providing the possibility of getting one-shot-revision decisions in lieu of rejection. To see a detailed explanation of the expected benefits from these changes, see "Additional Information about Multiple Deadlines Process" at www.usenix.org/conference/nsdi19/additional-info.

Two Deadlines

NSDI '19 offers authors the choice of two submission deadlines. Any paper submitted to one of these deadlines and accepted during the subsequent reviewing period will be presented at the conference and will appear as part of the proceedings. In the meantime, authors are permitted to advertise their papers as accepted by NSDI, for example listing them on CVs.

A paper submitted and rejected may not be submitted again to NSDI (even in revised form) until 11 months after the deadline it was submitted to.

One-Shot-Revision

Each paper may be accepted, rejected, or given the option of one-shot-revision. Such a revision decision includes a summary of the paper's merits and a list of necessary changes that are required for the paper to be accepted at NSDI. Authors may then submit a version of their work addressing those needs during the subsequent deadline. At that point, the paper will be reviewed to judge whether it addresses the requirements requested; this review will be conducted, to the extent possible, by the same reviewers as earlier. To enable this, PC members who give one-shot-revision decisions late in a year are obligated to participate as external reviewers in the following year to review those papers' resubmissions, which would be considered for the following year's conference. Papers revised and re-submitted following a one-shot-revision decision can only receive a decision of accept or reject, not revise; this is what makes revisions "one-shot."

The judgment about whether to accept a revised paper will be made as follows. Reviewers will primarily judge whether the authors have satisfied the requests accompanying the revision decision. They will also judge the resubmission on its independent merits, but should avoid rejecting it for non-fatal concerns that they could have raised during the first round of reviews. The reviewers should also ensure that the revised paper doesn't introduce new assertions without sufficient support. Unlike the shepherding process, the requested action points may include running additional experiments that obtain specific results, e.g., comparing performance against a certain alternative and beating it by at least 10%.

During the revision period, the paper is still considered under review to NSDI and therefore cannot be submitted to other conferences unless the authors first withdraw it from consideration. To make this obligation clear, authors who receive a one-shot-revision notification must, within two weeks of the notification, explicitly send an email acknowledging their participation in the one-shot-revision process. That email should indicate they understand that this means the USENIX Submission Policy (www.usenix.org/conferences/author-resources/submissions-policy) precludes concurrent submission to other conferences.

To make a one-shot-revision decision, reviewers must be comfortable accepting the paper if the authors make all the changes requested in it. Most notably, if a paper makes an insufficient contribution, or is incremental, then it should be rejected, not given a one-shot-revision decision. After all, the point of one-shot revision is not to produce highly-polished uninteresting papers, but rather to allow publication of exciting work that's unfortunately submitted in a form that's flawed in a way that can't be fixed with mere shepherding.

Reviewers will also be instructed to not offer a one-shot-revision option if they can't determine that the paper is adequate modulo the proposed revisions. For instance, if the paper is written so sloppily that there may be a hidden deep flaw, then the paper should be rejected, not given a one-shot-revision request to fix the inadequate writing.

Authors given a one-shot-revision decision will be sent, within a few days of the decision, detailed instructions about how to re-submit. These instructions will include the list of necessary changes that are required for the paper to be accepted. They will also explain how the authors should accompany their re-submission with auxiliary material to demonstrate how they've satisfied that list of changes. This auxiliary material will consist of (1) an additional version of the re-submission in which revision changes since the first submission are clearly marked, and (2) a separate textual explanation of the high-level differences between the two versions.

If authors receive a one-shot-revision decision but don't want to submit a revised version, they may withdraw it. In this case, they may not submit the paper to NSDI again until 11 months after the deadline they originally it submitted to.

If authors receive a one-shot-revision decision for a paper submitted to the fall deadline of NSDI '19, this gives them the option to make the requested changes and re-submit it to the next NSDI deadline, which is the first deadline of NSDI '20. If the paper is accepted then, it will appear at NSDI '20, not NSDI '19.

Operational Systems Track

In addition to papers that describe original research, NSDI '19 also solicits papers that describe the design, implementation, analysis, and experience with large-scale, operational systems and networks. We encourage submission of papers that disprove or strengthen existing assumptions, deepen the understanding of existing problems, and validate known techniques at scales or environments in which they were never used or tested before. Such operational papers need not present new ideas or results to be accepted; indeed, new ideas or results will not influence whether the papers are accepted. Note that the rules regarding submission and anonymization are different for operational systems track papers. Since the evaluation of operational systems track papers requires understanding the real-world use of the system, papers in this track will be reviewed in a more limited double-blind process. Authors' names should be withheld, as usual. However, in contrast to other papers, authors need not anonymize the content of their submission in any other way—they may keep company names, links, real system names, etc. as appropriate for the paper. Please note that you cannot switch tracks for your paper after submission since the submission rules differ.

Authors should indicate on the title page of the paper and in the submission form that they are submitting to this track.

The final program will make no distinction between papers accepted from this track and papers accepted from the regular track.

What to Submit

NSDI '19 is double-blind, meaning that authors should make a good faith effort to anonymize papers. Note that the operational track papers have different rules as described above. As an author, you should not identify yourself in the paper either explicitly or by implication (e.g., through the references or acknowledgments). However, only non-destructive anonymization is required. For example, system names may be left deanonymized, if the system name is important for a reviewer to be able to evaluate the work. For example, a paper on experiences with the design of .NET should not be re-written to be about "an anonymous but widely used commercial distributed systems platform."

Additionally, please take the following steps when preparing your submission:

- Remove authors' names and affiliations from the title page.
- Remove acknowledgment of identifying names and funding sources.
- Do not provide links to your own online content. If this online content is critical to the content of your paper, please see the submission form, which allows for some forms of content upload, or contact the PC chairs.
- Use care in naming your files. Source file names, e.g., Joe.Smith.dvi, are often embedded in the final output as readily accessible comments.
- Use care in referring to related work, particularly your own. Do not omit references to provide anonymity, as this leaves the reviewer unable to grasp the context. Instead, a good solution is to reference your past work in the third person, just as you would any other piece of related work. If you cite anonymous work, you will need to enter the deanonymized reference(s) on the online submission form.
- If you need to reference another submission at NSDI '19 on a related topic, reference it as follows: "A related paper describes the design and implementation of our compiler [Anonymous 2019]." with the corresponding citation: "[Anonymous 2019] Under submission. Details omitted for double-blind reviewing."
- Work that extends an author's previous workshop paper is welcome, but the paper should (a) acknowledge their own previous workshop publications with an anonymous citation and (b) explain the differences between the NSDI submission and the prior workshop paper. The online submission form will also require authors to submit the deanonymized citation and a short explanation of the differences from the prior workshop paper.
- Blinding is not intended to be a great burden. If blinding your paper seems too burdensome, please contact the program co-chairs and discuss your specific situation.

Submissions—as well as final papers—must be no longer than 12 pages, including footnotes, figures, and tables. Submissions may include as many additional pages as needed for references and for supplementary material in appendices. The paper should stand alone without the supplementary material, but authors may use this space for content that may be of interest to some readers but is peripheral to the main technical contributions of the paper. Note that members of the program committee are free to not read this material when reviewing the paper.

New in 2019: Submissions must be in two-column format, using 10-point type on 12-point (single-spaced) leading, in a text block **7" wide x 9" deep, with .33" inter-column space,** formatted for 8.5" x 11" paper. Please note that the text block size has changed.

Papers not meeting these criteria will be rejected without review, and no deadline extensions will be granted for reformatting. Pages should be numbered, and figures and tables should be legible when printed without requiring magnification. Authors may use color in their figures, but the figures should be readable when printed in black and white. If you wish, you may use the template for LaTeX available on the conference paper templates page at www.usenix.org/conferences/author-resources/paper-templates. All papers must be submitted via the submission form linked from the NSDI '19 Call for Papers web page. Please do not email submissions.

Submissions will be judged on originality, significance, interest, clarity, relevance, and correctness.

Policies

Simultaneous submission of the same work to multiple venues, submission of previously published work, or plagiarism constitutes dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may take action against authors who have committed them. See the USENIX Conference Submissions Policy at www.usenix.org/conferences/author-resources/submissions-policy for details.

Previous publication at a workshop is acceptable as long as the NSDI submission includes substantial new material that has been developed since the publication of any earlier version. However, NSDI submissions cannot be concurrent with submission to a workshop venue. If the notification date for the workshop submission is after the submission date for NSDI (as is the case for ACM HotNets 2018), this would be considered a concurrent submission and would be rejected without review. Such concurrent submissions would have limited the possibility of substantially extending the prior work, which would violate the intent of policies allowing for extended submissions (as described in www.sigcomm.org/about/policies/frequently-asked-questions-faq/) See remarks above about how to cite and contrast with a workshop paper.

Authors uncertain whether their submission meets USENIX's guidelines should contact the Program Co-Chairs, nsdi19chairs@usenix.org.

Papers accompanied by nondisclosure agreement forms will not be considered. All submissions will be treated as confidential prior to publication on the USENIX NSDI '19 website; rejected submissions will be permanently treated as confidential.

Ethical Considerations

Papers describing experiments with users or user data (e.g., network traffic, passwords, social network information), should follow the basic principles of ethical research, e.g., beneficence (maximizing the benefits to an individual or to society while minimizing harm to the individual), minimal risk (appropriateness of the risk versus benefit ratio), voluntary consent, respect for privacy, and limited deception. When appropriate, authors are encouraged to include a subsection describing these issues. Authors may want to consult the Menlo Report at www.caida.org/publications/papers/2012/menlo_report_actual_formatted/ for further information on ethical principles, or the Allman/Paxson IMC '07 paper at http://conferences.sigcomm.org/imc/2007/papers/imc76.pdf for guidance on ethical data sharing.

Authors must, as part of the submission process, attest that their work complies with all applicable ethical standards of their home institution(s), including, but not limited to privacy policies and policies on experiments involving humans. Note that submitting research for approval by one's institution's ethics review body is necessary, but not sufficient—in cases where the PC has concerns about the ethics of the work in a submission, the PC will have its own discussion of the ethics of that work. The PC's review process may examine the ethical soundness of the paper just as it examines the technical soundness.

Processes for Accepted Papers

If your paper is accepted and you need an invitation letter to apply for a visa to attend the conference, please contact conference@usenix.org as soon as possible. (Visa applications can take at least 30 working days to process.) Please identify yourself as a presenter and include your mailing address in your email.

Accepted papers may be shepherded through an editorial review process by a member of the Program Committee. Based on initial feedback from the Program Committee, authors of shepherded papers will submit an editorial revision of their paper to their Program Committee shepherd. The shepherd will review the paper and give the author additional comments. All authors, shepherded or not, will upload their final file to the submissions system by the camera ready date for the conference Proceedings.

Paper publishing schedule: A list of papers accepted from the Spring submissions will be posted on the NSDI '19 website in August. In December, when the full program is available, paper titles and abstracts will be posted for all accepted papers from both the Spring and Fall deadlines. At this time, the Spring final paper PDFs will also be posted, accessible only to registered attendees. In February, the full Proceedings as well as all of the final paper PDFs will be posted.

All papers will be available online to registered attendees before the conference. If your accepted paper should not be published prior to the event, please notify production@usenix.org. The papers will be available online to everyone beginning on the first day of the conference.

Best Paper Awards

Awards will be given for the best paper(s) at the conference.

Community Award

To encourage broader code and data sharing within the NSDI community, the conference will also present a "Community Award" for the best paper whose code and/or data set is made publicly available by the final papers deadline, February 13, 2019. Authors who would like their paper to be considered for this award will have the opportunity to tag their paper during the submission process.

Rev. 4/27/18

27TH USENIX SECURITY SYMPOSIUM

BALTIMORE, MD, USA

Co-located Workshops

WOOT '18

12th USENIX Workshop on Offensive Technologies
August 13–14, 2018
www.usenix.org/woot18

WOOT presents a broad picture of offense and its contributions. Offensive security today is a large-scale operation managed by organized, capitalized actors, and software used by millions is built by startups less than a year old. In the field's infancy, offensive security research was conducted separately by industry, independent hackers, or in academia, and collaboration between these groups could be difficult. Since 2007, WOOT has brought these communities together to share, explore, and produce high-quality, peer-reviewed work discussing tools and techniques for attack.

ASE '18

2018 USENIX Workshop on Advances in Security Education
August 13, 2018
www.usenix.org/ase18

Educators, designers, and evaluators attend ASE to collaborate, share cutting-edge research in computer security education, improve existing practices, and validate or refute widely held beliefs within the field. The workshop program covers computer security education in various settings and with a diverse set of goals, including developing or maturing specific knowledge, skills, and abilities, and improving awareness of issues in the cyber domain.

CSET '18

11th USENIX Workshop on Cyber Security Experimentation and Test
August 13, 2018
www.usenix.org/cset18

CSET is a forum for researchers and practitioners in academia, government, and industry to explore the significant challenges within the science of cyber security. Presenters and attendees are encouraged to engage in interactive discussions on cyber security evaluation, experimentation, measurement, metrics, data, simulations, and testbeds for software, hardware, or malware.

FOCI '18

8th USENIX Workshop on Free and Open Communications on the Internet
August 14, 2018
www.usenix.org/foci18

Political and social change around the world is driven by Internet communications, which governments and other actors seek to control, monitor, and block using multifarious methods. These threats to free and open communications on the Internet raise a wide range of research and interdisciplinary challenges. FOCI brings together researchers and practitioners from technology, law, and policy who are working on means to study, detect, or circumvent practices that inhibit free and open communications on the Internet.

HotSec '18

2018 USENIX Summit on Hot Topics in Security
August 14, 2018
www.usenix.org/hotsec18

Researchers across computer security disciplines convene at HotSec to discuss the state-of-the-art, with emphasis on future directions and emerging areas. HotSec is not your traditional security workshop! It's a series of lightning talks sessions on emerging work and positions in security, followed by discussion among attendees. The format provides a quick and informal way to share ideas and inspire breakout discussions for the remainder of the day.

Register by July 23 and save!

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES