;login:

DECEMBER 2015 VOL. 40, NO. 6

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Security

CAN bus
Ian Foster and Karl Koscher

The Future of Digital Forensics
Simson L. Garfinkel

The Failure of Megamos Crypto
Roel Verdult and Flavio D. Garcia

How NVM Changes Everything: Interview with Darrell Long
Rik Farrow

History
Peter H. Salus on STUG and LISA
Interview with Rob Kolstad

Columns
Python 3.5s and the new Async-IO
David Beazley

Practical Perl Tools
David N. Blank-Edelman

Proper Metrics for Logging
Dave Josephsen

The Unpredictability of the Future
Dan Geer

usenix 40TH ANNIVERSARY

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

Enigma 2016
January 25–27, 2016, San Francisco, CA, USA
enigma.usenix.org

FAST '16: 14th USENIX Conference on File and Storage Technologies
February 22–25, 2016, Santa Clara, CA, USA
www.usenix.org/fast16

NSDI '16: 13th USENIX Symposium on Networked Systems Design and Implementation
March 16–18, 2016, Santa Clara, CA, USA
www.usenix.org/nsdi16

Co-located with NSDI '16

CoolDC '16: USENIX Workshop on Cool Topics on Sustainable Data Centers
March 19, 2016
Submissions due December 15, 2015
www.usenix.org/cooldc16

SREcon16
April 7–8, 2016, Santa Clara, CA, USA

USENIX ATC '16: 2016 USENIX Annual Technical Conference
June 22–24, 2016, Denver, CO, USA
Submissions due February 1, 2016
www.usenix.org/atc16

Co-located with USENIX ATC '16:

HotCloud '16: 8th USENIX Workshop on Hot Topics in Cloud Computing
June 20–21, 2016
Submissions due March 8, 2016
www.usenix.org/hotcloud16

HotStorage '16: 8th USENIX Workshop on Hot Topics in Storage and File Systems
June 20–21, 2016
Submissions due March 10, 2016
www.usenix.org/hotstorage16

SOUPS 2016: Twelfth Symposium on Usable Privacy and Security
June 22–24, 2016
Paper registration due March 1, 2016
www.usenix.org/soups2016

SREcon16 Europe
July 11–13, 2016, Dublin, Ireland

USENIX Security '16: 25th USENIX Security Symposium
August 10–12, 2016, Austin, TX, USA
Submissions due February 18, 2016
www.usenix.org/sec16

Co-located with USENIX Security '16

WOOT '16: 10th USENIX Workshop on Offensive Technologies
August 8–9, 2016

CSET '16: 9th Workshop on Cyber Security Experimentation and Test
August 8, 2016

ASE '16: 2016 USENIX Workshop on Advances in Security Education
August 9, 2016

HotSec '16: 2016 USENIX Summit on Hot Topics in Security
August 9, 2016
www.usenix.org/hotsec16

OSDI '16: 12th USENIX Symposium on Operating Systems Design and Implementation
November 2–4, 2016, Savannah, GA, USA
Abstracts due May 3, 2016
www.usenix.org/osdi16

LISA16
December 4–9, 2016, Boston, MA, USA
www.usenix.org/lisa16

Do you know about the 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

Stay Connected...

twitter.com/usenix
www.usenix.org/facebook
www.usenix.org/youtube
www.usenix.org/linkedin
www.usenix.org/gplus
www.usenix.org/blog

;login:

DECEMBER 2015 VOL. 40, NO. 6

EDITORIAL

2 Musings *Rik Farrow*

SECURITY

6 Exploring Controller Area Networks *Ian Foster and Karl Koscher*

12 The Expanding World of Digital Forensics *Simson L. Garfinkel*

17 Cryptanalysis of the Megamos Crypto Automotive Immobilizer *Roel Verdult and Flavio D. Garcia*

23 Ethical Behavior in Cyberspace Research *John Murray*

26 Hack, Play, Win: Lessons Learned Running the Maryland Cyber Challenge *Richard Forno*

SYSADMIN

32 /var/log/manager: I'm the Manager, This Is My Job *Andy Seely*

FILE SYSTEMS

36 Interview with Darrell Long *Rik Farrow*

HISTORY

38 Offshoots: STUG and LISA *Peter H. Salus*

40 Interview with Rob Kolstad *Rik Farrow*

43 *;login:* The UNIX Newsletter

COLUMNS

48 Awaiting for Godot *David Beazley*

55 Practical Perl Tools: OAuth2 in Situ *David N. Blank-Edelman*

60 iVoyeur: Tests and Metrics *Dave Josephsen*

64 For Good Measure: Why Speculate? *Dan Geer*

66 /dev/random: Streamailer and Beyond *Robert G. Ferrell*

BOOKS

68 Book Reviews *Mark Lamourine and Rik Farrow*

USENIX NOTES

70 2016 Election for the USENIX Board of Directors *Casey Henderson*

71 Team USA Continues to Impress at IOI 2015 in Kazakhstan *Brian C. Dean*

72 Thanks to Our Volunteers *Casey Henderson*

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 bi-monthly 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 non-members are $90 per year. Periodicals postage paid at Berkeley, CA, and additional offices.

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

usenix 40TH ANNIVERSARY 1975–2015

EDITORIAL

Musings

RIK FARROW

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

I've often written about how depressing I find computer security is for the December issue, so this year I thought I'd try a different tack. Honestly, there were parts of USENIX Security, particularly the WOOT workshop, that had me laughing out loud.

I really liked the "Fast and Vulnerable" [1] paper for its humorous insights into the state of programming. A widely used product, one that is Internet-connected and can be used to control cars, totally fails at having any security at all. What a laugh! They even included the private SSH key for the root account for the device—and the same key is used on all devices by this manufacturer.

Not that SSH is needed at all: just a simple SMS message to the device can be used to instruct it to download a software update. That's right. All you need is a phone number and to send a text message, and you can "own" someone else's car. And the phone number could be war-dialed. As if this weren't enough, there's also a Web *and* a Telnet interface you can use.

Programmers

I've heard Wietse Venema and D. J. Bernstein's names mentioned many times at USENIX Security as the only people who have a proven track record for writing secure software. That should give you food for thought: if just these two guys have done it right, that implies everyone else is doing it wrong.

And that seems to be about right. Getting the security right is very hard; even the best programmers often make exploitable mistakes, and I think we should assume that the best programmers are a tiny minority. That leaves about six nines, or 99.9999% of programmers among those who are not the best. Then what those statements about Venema/Bernstein really mean is that effectively no one can write software securely.

A large part of the problem has to do with the nature of programming. Someone, hopefully a skilled programmer, gets tasked with creating software that will convert requests into the appropriate responses. In most cases, the programmer writes a list of instructions, does some testing, then keeps working on the software until it appears to work.

Nowhere in that outline of the programming task does the concept of security even appear. If security does come up, the requirement is often something like "must include crypto," and the programmer then adds a function with a hash check or perhaps XORs something so the program includes encryption. Perhaps the programmers are more advanced and decide they will use a library in OpenSSL, but because they don't understand cryptography they use weak keys and repeat the same initialization vector with every request.

I used to laugh at early attempts to port Microsoft MS-DOS programs to UNIX. The authors didn't understand the UNIX security model, so they would run their software as root. All files would be publicly writeable by all, too. For someone coming from the MS-DOS world, where there was no security, this was the equivalent security "ported" to UNIX.

In the research for "Fast and Vulnerable," the programmers/designers for the car-connected device didn't do much better. They did have a root password and a user password (for the user named "user"), but they were trivially cracked. You can query, update, or control the device either locally, over a network connection (via a USB port), or remotely, as the Web and Telnet services appear both locally and remotely. And since this device connects to the OBD-II port on US-built cars, you can play the types of tricks with the car as Miller and Valasek did with the entertainment head found in new Chrysler Jeeps [2].

The Question

Finding vulnerable devices online is nothing new. Companies have created everything from routers to medical devices that appear online, have software with exploitable vulnerabilities, and provide no mechanism for updating the software of these devices, and this has been true for about as long as there has been a public Internet (1991). You might think that we, the computer science community, might try and do something about this sorry state of affairs. And we have, but because security isn't easy, things haven't worked out so well.

Let's take a look at a couple of recent attempts to provide security for applications that assume that programmers shouldn't be expected to do this themselves: Android and iOS. Of the two, iOS has a stricter model, which has worked well until XcodeGhost [3] came along. By adding a Trojan object file to the Xcode development framework, used to write applications for iOS, applications would include the Trojan binary. And Apple accepted these applications into the Apple App Store, as the applications appeared to follow the rules. Ooops.

Android has fared much worse for a couple of reasons. Google never wanted, or could have, the same degree of control over the apps market for Android, and that has opened the field to malware. And second, the model chosen for Android was never intended to be as robust. When I first heard about the Android security model, at a USENIX Security Symposium in Montreal (2009), I heard that the model would partially rely on people noticing and complaining about insecure apps. Also, users would be expected to decide whether or not to allow apps access to their devices, with over 140 different types of access potentially allowed. I made a point of speaking with the program manager, and when I complained to him that they expected way too much from Android users, he told me that it was too late to change the design.

Google has recently updated the Android security design [4], but the user still must make security decisions, and those decisions are still all or nothing. For example, if you want to use the Uber app, you grant Uber complete access to your phone and personal information. I found that very interesting. I also know that lots of people are willing to trust Uber, the corporation, and Uber programmers, with complete access to their phones. Really? Have you done that?

Google has created a new sandbox, without any access permissions, that apps can be run in. Another of my favorite papers at Security, "Boxify: Full-Fledged App Sandboxing for Stock Android" [5], will allow knowledgeable programmers to run other people's apps inside a permissionless sandbox, and have finer-grained control over what personal data we are willing to share and when. What distinguishes the new sandbox from the old one is the ability to run unmodified apps, regardless of what permissions the app programmers have asked for (everything if you're Uber), and decide to grant access to a selected set when executed, instead of at install time.

Microsoft got serious about security in 2001. Just this year, they replaced the default browser, IE, with something better. Apple and Google have always been serious about security, but their results have been mixed so far. Keep in mind that iOS in China, where lots of iPhones are jailbroken, is on a par with Android in the rest of the world. None of this is easy.

To sum up, we have a history of programmers being unable to write programs securely. We have vendors who are unable to provide secure environments in which to run insecure apps. So why are we at all surprised at the lack of security today?

The Lineup

I was so impressed with Ian Foster's WOOT '15 presentation about finding weaknesses in an OBD II device that I asked him and one of his co-authors (Karl Koscher) if they would write about CAN bus for *;login:*. Understanding CAN bus is the key to understanding how modern cars work—and why we see remote hacks of cars that might appear to be magic if we didn't know better.

Roel Verdult and Flavio Garcia explain the problems found in Megamos, a widely used automobile immobilizer. In what could be seen as a reprise to the theme of my editorial, Verdult and Garcia analyzed both the immobilizer itself *and* how it has been used in many brands of automobiles, showing that indeed, programmers cannot program securely and vendors don't understand cryptography.

Simson Garfinkel volunteered an article on digital forensics. Fresh from chairing the DFRWS 2015 conference, Simson provides us with a clear view of how the field of digital forensics has grown both broader and more challenging over time. Both a current researcher and an accomplished writer, Simson takes us through from the early days of cloning drives to modern tools that can analyze the vast amount of data found on our digital devices and networks.

Musings

I met John Murray during the USENIX Security '15 poster session, where he was explaining the Menlo Report to anyone who would listen. I'd only that afternoon heard of the Menlo Report, during a panel session by past program committee members discussing how they handle ethical lapses in submitted papers. I asked John if he could tell us more about how the Menlo Report provides a better basis for institutional review boards (IRBs) considering security research proposals.

I've known Rick Forno for many years, from when he worked for the early US Internet name registry. Rick offered to share five years of experience running the Maryland Cyber Challenge, a cyber competition that allows participants at different levels of education and experience to learn together.

Andy Seely has written a summary of his 11 columns on managing system administrators. If you missed any of his past columns, you can review the useful practices that Andy has shared, as well as locate the columns you either missed or find that you now need.

I interviewed Darrell Long (UCSC) about how the actual arrival of a real non-volatile memory (NVM) RAM product will affect the world of computing. The answer? You need to read the interview, but I can tell you here, this is big.

Peter Salus writes his final USENIX history article and discusses the founding of the Software Tools User Group (STUG) and LISA. If you've ever wondered where USENIX came from, who came up with that name, I suggest you read all of Peter's articles. They aren't that long but do put the past of USENIX in perspective.

I also interview Rob Kolstad. Rob was elected to the USENIX Board three times and started the LISA conference. Rob is an amazing guy, and you can find hints of this by reading this interview, as well as another look at where USENIX has come from and what happened in the past to make it the organization that it is today.

Dave Beazley waxes enthusiastic about the new asynchronous I/O (asyncio) module in Python 3.5. As Dave writes, this will blow your mind if you are already familiar with past asyncio Python modules. More practically, you will learn the current direction for handling thousands of threads of activity using coroutines.

David Blank-Edelman took up the challenge I tossed him when he submitted his October column and worked up some Perl script magic for using OAuth2. OAuth2 is more than an authentication protocol, as OAuth2 tokens are used for delegating access to resources on other folks' servers. David's column covers checking all of the Google Calendars shared with you for events that may be bugging you with daily notifications, perhaps while the person involved is off on vacation.

Dave Josephsen discusses monitoring for programmers. While we usually think of monitoring as a task belonging to system administrators and SREs, Dave reminds us that developers need to understand what types of events they should be reporting in their code.

Robert Ferrell uses his fertile imagination to come up with a new form of secure email, Streamailer, and also addresses how difficult change in *any* form is for many people, and ends with a new technique for authentication.

We have book reviews by both Mark Lamourine and myself. Mark reviewed *Python for Data Analysis* and an introductory book on Go, while I reviewed the Donovan and Kernighan Go book.

Restarting

I believe that if people are allowed to do anything, including things harmful to themselves or others, someone will try to do those harmful things. And it appears that humanity in general agrees with this belief, as we have laws and customs that set constraints on behavior. We drive on a particular side of roads, do not pick up apples at a farmer's market and walk off with them without paying, throw stones or shoot at passersby, and so on. We live in a civil society (for the most part).

But in the world of programming, we have few constraints. The "Fast and Vulnerable" [1] paper clearly shows what happens when programmers are set free of the constraints of security to design a product that, when misused, can kill people.

I also believe that people are much more comfortable with constraints that appear natural. We all learn as children that when we jump up, we quickly fall back to earth, if we eat too much, our stomach starts hurting, and so on. These are natural constraints, and they work for the most part (see 72-ounce steak rules [6]), for most people.

That said, I also believe that we need programming environments where writing secure code comes naturally. I believe we can do that with modern languages and a structure of natural constraints that encourage writing small modules that require least privileges, as Venema and Bernstein have been showing us for years.

When we, the culture of computer scientists, began writing programming languages, we needed something better than writing in machine language, and we got FORTRAN and COBOL. We've come a long way since then, but it's painfully obvious we still have a long way to go.

There is another important leg to having an environment where secure programming feels natural, and that is the hard part: hardware. Our programming environment matches our hardware architecture, where even a smartwatch has an architecture that evolved from 1960 mainframes. We are still building time-sharing systems, even now that we can put tens of cores on a tiny chip. Those cores, properly connected, could form the basis for many small services that work today.

There's even a name for those services that has become quite popular: microservices. Let's build the architecture [7] for running those microservices natively and securely, and design a software architecture that makes programming securely natural.

References

[1] I. Foster, A. Prudhomme, K. Koscher, S. Savage, "Fast and Vulnerable: A Story of Telematic Failures," in the *Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT '15)*, August 2015, Washington, DC.

[2] Andy Greenbaum, "Hackers Remotely Kill a Jeep on the Highway—With Me in It," July 21, 2015: http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[3] http://researchcenter.paloaltonetworks.com/2015/09/novel-malware-xcodeghost-modifies-xcode-infects-apple-ios-apps-and-hits-app-store/.

[4] Security-Enhanced Linux in Android: https://source.android.com/devices/tech/security/selinux/index.html.

[5] Michael Backes, Sven Bugiel, Christian Hammer, Oliver Schranz, and Philipp von Styp-Rekowsky, "Boxify: Full-Fledged App Sandboxing for Stock Android," in the *Proceedings of the 24th USENIX Security Symposium*, August 2015, Washington, DC, pp. 691–706: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/backes.

[6] The 72 oz. Steak Rules: http://bigtexan.com/72oz-steak-rules/.

[7] http://www.rikfarrow.com, Design for Security.

Exploring Controller Area Networks

IAN FOSTER AND KARL KOSCHER

Ian Foster recently completed his master's degree at the University of California, San Diego, where he worked on analyzing the security of aftermarket telematics dongles. In doing so he found that security of some of these devices was often an afterthought, if existent at all. idfoster@cs.ucsd.edu

Karl Koscher is a postdoctoral researcher at the University of California, San Diego, where he specializes in embedded systems security. He earned his PhD in 2014 from the University of Washington, working with Tadayoshi Kohno. As part of his dissertation work, he was one of the lead researchers to perform the first published, comprehensive, experimentally backed security analysis of a modern automobile. supersat@cs.ucsd.edu

The highly publicized attack by Miller and Valasek during the summer of 2015 once again drew attention to weaknesses in automobile security. All modern automobiles rely on a broadcast network called CAN, and interfaces into that network are actually required by law. In this article, we explain how the CAN bus works and how it can be exploited.

Background

The Controller Area Network (CAN) is a serial bus standard designed for reliable, real-time message delivery between distributed control systems. Originally intended for vehicle applications, the CAN bus standard has found its way into many types of control systems, such as those used in elevators, medical devices, and robots. As detailed below, the standard is commonly implemented as a shared, differentially signaled bus, and enables priority-based arbitration. Multiple bitrates are supported, up to one megabit per second.

In automotive contexts, CAN buses are now commonly used to connect the various computers (known in the industry as electronic control units, or ECUs) of a car together. These ECUs now control most aspects of modern automobiles, including the engine, transmission, brakes, airbags, lights, and locks. Additional systems, such as "infotainment" (e.g., radio/navigation systems) and telematics systems (e.g., OnStar), are often connected to these ECUs. Vehicles will often have multiple CAN buses connecting various subsets of ECUs together.

The Controller Area Network Standard

Bosch, a German manufacturer of automotive control systems, began work on the Controller Area Network standard in 1983. Intel and Mercedes-Benz became involved with the project shortly thereafter, and in 1986 a paper introducing the "Automotive Serial Controller Area Network" standard was presented at the annual International Congress of the Society of Automotive Engineers (SAE) [1]. Version 2 was released in 1991 and forms the basis of all modern CAN implementations. CAN was subsequently adopted as an ISO standard (11898) in 1993 [2].

The CAN standard is optimized for low latency, high throughput, and reliable transmission. Low latency is achieved through short frame sizes (with a maximum payload length of eight bytes) and a priority-based, carrier sense multiple access (CSMA) arbitration scheme. While the maximum bitrate of 1 Mbps may seem low by today's standards, it meets the needs of most control systems. A new, backwards-compatible extension called CAN FD supports higher data rates. Reliability is ensured through multiple mechanisms. Differential signaling is commonly used at the physical layer, which provides immunity to common-mode noise (i.e., interference that couples onto one line will couple on to the other as well, canceling out its effect), as well as potential redundancy if one of the lines should fail. A 15-bit CRC (cyclic redundancy check) field at the end of each frame provides a high amount of certainty that frames are received correctly and are uncorrupted. An ACK slot at the end of the frame allows the sender to ensure that the frame was received correctly by at least one node, and many CAN controllers will automatically retransmit unacknowledged CAN frames. Figure 1 shows the CAN frame format.

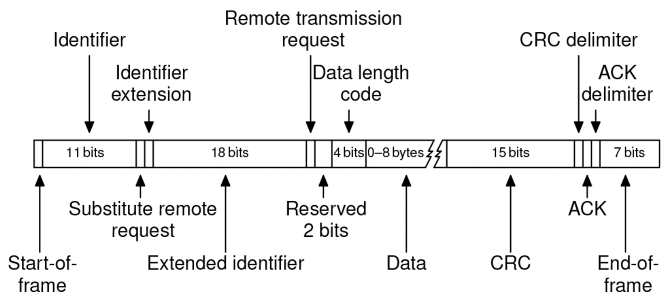


Figure 1: The CAN frame format

The CAN standard allows some flexibility in the physical layer (and in fact is not specified in Bosch's original standards), but relies on there being a "dominant" logical zero state and a "recessive" logical one state. Most CAN applications implement the physical layer described in ISO 11898-2, which specifies two lines, CAN_H and CAN_L, which are connected to each other at both ends of the bus with a 120Ω terminating resistor. In the recessive state, CAN_H and CAN_L are at approximately the same voltage (nominally 2.5v to ground). In the dominant state, CAN_H is pulled to 5v, while CAN_L is pulled to ground.

CAN frames primarily consist of an 11-bit or 29-bit arbitration ID, a four-bit length field, up to eight bytes of payload, a 15-bit CRC, and an acknowledgment bit. The arbitration ID typically is treated as a message type, but is sometimes (such as with OBD-II diagnostics, described below) used as a controller source or destination identifier. When transmitting the arbitration ID, the CAN transceiver monitors the bus. If it sends a recessive bit but detects a dominant bus condition, it aborts the message transmission. Note that the frame from the node that asserted the dominant bus condition has not been corrupted and thus can continue to be sent. Since a dominant state indicates a logical zero, and data is transmitted most-significant-bit first, lower arbitration IDs take precedence over higher arbitration IDs.

At this point, the astute reader may notice some security issues with the CAN protocol as described. In particular, CAN buses are broadcast networks, typically don't provide a way to identify the sender or recipient of a message, and are subject to trivial denial-of-service attacks. Each node on a CAN bus can observe all traffic. In fact, aspects of the CAN protocol, such as arbitration, *require* this. Furthermore, each node can send arbitrary CAN frames, without other nodes being able to verify the sender. Source or destination IDs, if used, can be trivially spoofed. Constantly asserting a dominant bus state will cause all other nodes to back off indefinitely, although well-designed CAN transceivers will detect this and enter a receive-only mode, making this type of denial-of-service attack difficult to pull off in software alone.

Given that the CAN standard provides no protection against malicious behavior, an attacker with access to a CAN bus is often able to take control of many critical aspects of the attached control systems. In the case of modern automobiles, there are many potential entry points into a vehicle's CAN bus(es), and these buses expose almost complete control over every aspect of the car's operation.

CAN Buses in Vehicles

Since CAN was invented with automotive applications in mind, we should step back and explain why vehicle ECUs may want to communicate with each other. Early engine control systems were introduced to meet stringent new emissions limits. In particular, by monitoring multiple sensors, the air/fuel ratio could be tightly controlled to minimize emissions. Since then, ECUs have evolved and proliferated to support ever-increasing fuel efficiency, emissions, safety, and reliability standards, and there can be further synergies with cross-ECU communications. For example, as Bosch explained in their original CAN bus paper, a transmission control unit can request the engine control unit to reduce torque, which reduces wear on the clutch and provides smoother shifting [1]. Similarly, the airbag controller can ask the engine controller to shut off the fuel pump if the airbags deploy, minimizing the chance of a fuel leak after an accident. Faced with a growing amount of inter-ECU communication, moving to a shared communications bus reduced the number of expensive (and heavy) point-to-point links.

Today, most aspects of a vehicle's operation go over one or more CAN buses. For example, in one vehicle we looked at [3], the anti-lock braking/stability control system reports the vehicle's speed to other modules, such as the speedometer, as well as to the engine controller (as input to its cruise-control algorithms). The radio also receives these speed messages to dynamically adjust its volume. In fact, the familiar click-clacks of the turn signal relays are now simulated by the radio, which receives the turn signal status from the body controller. The telematics system routes its audio through the radio, and can command the HVAC system to turn down the fans when a call is received.

These are just a few examples of inter-ECU communication in today's modern vehicles. In fact, the amount of information being transferred has grown to a point where vehicles often have *multiple* CAN buses, with gateway nodes that route selected messages between these buses. The architecture of how ECUs are connected together varies a great deal by manufacturer, but in 2014 Miller and Valasek published a survey of CAN bus architectures across a wide range of OEMs [4].

On-Board Diagnostics

In 1996, the OBD-II (On-Board Diagnostics) connector became federally mandated by the US Government. The OBD-II connector provides a way to verify the status of emissions control systems and facilities emissions testing. For example, emissions control systems can indicate over OBD-II port whether any sensor faults have been detected, the overall confidence in sensor performance, whether any unapproved firmware modifications have been made (which may affect emissions), as well as current sensor readings, which can be validated against external testing equipment. At the time, while the physical connector was standardized, there were several OBD-II communication protocols used by different manufacturers. The widespread adoption of CAN for powertrain ECU communications led it to be the natural choice for a single OBD standard. Since 2008, all vehicles sold in the US are required to provide OBD functionality over CAN. Practically speaking, this means that one or more major CAN buses are typically exposed to the OBD-II port.

The legally-required implementation of OBD over CAN ("legislated OBD") is defined by ISO 15031 and ISO 15765 and provides a relatively limited set of services, such as reading certain powertrain parameters such as engine speed, retrieving and clearing trouble codes, retrieving historical parameters recorded when a trouble code was raised, and requesting sensor test data. Under these standards, diagnostic messages are directed to ECUs at fixed CAN IDs, with their responses coming back with other fixed CAN IDs. ISO 15765-2 defines a simple transport layer, known as ISO-TP, which can be used to assemble larger diagnostic messages across multiple CAN frames and ensures in-order delivery [5].

In addition to legislated OBD, many vehicles also support the newer Unified Diagnostic Services (UDS) standard, defined by ISO 14229-3, which builds on legislated OBD. UDS provides several additional services, such as the ability to read and write arbitrary memory locations in ECUs, reflash ECU firmware, and override ECU I/O. For sensitive operations, such as reflashing safety, theft, or emissions-critical ECUs, or performing potentially unsafe I/O overrides, an OEM-defined unlocking process must usually be performed with the UDS SecurityAccess service, which defines a challenge/response-type mechanism for authentication. However, these unlocking schemes are often not robust—some OEMs use small keys that can be brute-forced, while others use simple algorithms such as XORing the challenge with a known secret [3, 6].

OBD-II Example

In the following example we show how to request the vehicle's VIN from the engine control module using OBD over CAN. OBD query and response packets are sent over the CAN bus using ISO-TP standard [5]. In this example, all nibbles and bytes shown are part of the CAN frame's eight-byte data section except for the ID field.

ID	Type	LEN	SID	PID
#0x7E0	0	2	09	02

Figure 2: OBD-II VIN query frame—only the ID field and eight-byte data field of the CAN frame are shown.

The ID or priority of the example CAN frame shown in Figure 2 is 0x7E0. CAN identifiers for legislated OBD are defined by ISO 16765-4 [7], which specifies that ECUs can be physically addressed with IDs 0x7E0–0x7E7, with corresponding replies sent to 0x7E8–0x7EF. The frame type and length are both nibbles. The type is 0x0 to indicate a "single frame," which means that the entire OBD request can fit within a single CAN frame. The LEN field specifies how many more bytes follow in the request. SID is the service identifier, which in this case is 0x09, or the "Request Vehicle Data" service. The "Request Vehicle Data" service takes a parameter ID (PID). For this service, a PID of 0x02 corresponds to the VIN.

ID	Type	LEN	SID	PID	NO	Data
#0x7E8	1	014	49	02	01	VIN[00-02]

Figure 3: OBD-II initial VIN response frame

The response to the query frame is shown in Figure 3. The ID and PID code fields should be the same as the query frame. As with the request, if the response can fit within a single frame, the type is 0. However, in this case, the response is split across many frames, so a frame type of 1 is used to indicate the "start frame" of a multi-frame packet. The LEN field of a start frame indicates the total number of bytes in the response. In an OBD response, the SID field is equal to 0x40 plus the SID from the query. For service 0x09, NO is the number of data items (in this case 1 for the VIN). Data contains the first three bytes of the requested data.

ID	Status	BS	ST
#0x7E0	30	00	00

Figure 4: ISO 15765-2 OBD-II flow control frame sent to main ECU

In order to get the remaining 17 bytes, a flow control frame needs to be sent to the ECU informing it of the parameters for sending consecutive frames. Figure 4 shows a flow control frame that will instruct the ECU to send all of the remaining packets immediately. The ID is the same as the OBD query. A status of 0x30 requests the rest of the data to be sent now and a status of 0x31 requests the ECU to wait. BS is the block size, defining the number of frames to send before waiting for next flow control frame (0 means no further flow control frames are needed). ST is the separation time delay between frames in milliseconds.

ID	Type	Index	Data
#0x7E8	2	1	VIN[03-09]
#0x7E8	2	2	VIN[10-16]
#0x7E8	2	3	VIN[17-19]

Figure 5: Remaining OBD-II VIN response frames

Figure 5 shows the remaining data frames sent by the ECU after receiving the flow control frame. The ID is the same as the initial response frame. Type and Index are both nibbles. The type is 0x2 to indicate consecutive frames, and the index is a frame counter starting at 1. Data contains up to seven bytes of the response data per consecutive frame. In this example, VIN is represented as a 20-byte string that is divided up into an initial frame and three consecutive frames.

This example uses a well-known query to request the VIN from the main ECU. However, for every ECU on the bus there are many other packets that are not well-documented. The more interesting CAN frames that can affect things like the engine, brakes, locks, etc. are proprietary and are generally not shared by the manufacturer.

Most of what is publicly known about the non-standard CAN frames has been reverse-engineered. Each vehicle may have different CAN messages, and sometimes even different generations of the same vehicle will use different frames. For example, the CAN frame to unlock the trunk on one vehicle may activate the windshield wipers of another vehicle.

Exploiting Vehicular Controller Area Networks

We now turn our attention towards how these automotive CAN buses can be abused. An attacker may be able to get access to these CAN buses in a variety of ways. Since these buses are often exposed over the ODB-II port, aftermarket devices that plug into this port (such as dongles that track your driving for insurance purposes) are potential entry points. At WOOT '15 we demonstrated several attacks against a popular OBD-II dongle platform that gives an attacker complete access to at least one CAN bus [8]. These dongles connect cars to the cellular network and can be exploited via SMS or their built-in Web server. Prior work by researchers at UC San Diego and the University of Washington, as well as Miller and Valasek, have also demonstrated *multiple remotely exploitable vulnerabilities in unmodified vehicles,* which can also be used to gain complete access to CAN buses [9, 10]. With vehicles becoming increasingly connected to the outside world, the number of potentially vulnerable entry points to these vehicles' CAN buses is rapidly growing.

With access to the CAN buses, an attacker can either use standard inter-ECU messages to influence vehicle behavior or may be able to exploit diagnostic services. For example, Miller and Valasek demonstrated partial control of the steering wheel by spoofing parking-assist and lane-keep-assist messages. These messages are relatively easy to discover—since the CAN bus is a broadcast network, we can simply monitor the messages sent while eliciting a behavior we want to reproduce. These messages can be captured using a CAN frame logger connected to the ODB-II port, and we can verify that we've found the correct message by replaying it and seeing if it produces the desired effect.

Given the relatively fragile nature of CAN, an attacker can override messages as well. For example, the UW/UCSD researchers were able to falsify speedometer readings—and in fact, invert them such that the displayed speed was 100 MPH *minus* the actual speed—simply by flooding the bus with spoofed messages [3]. A slightly more sophisticated attack could detect speedometer messages sent by other ECUs and assert a dominant bus state during the CRC, causing all other receivers to reject the message as invalid, although this requires fairly precise timing.

Some "functionality" is not exposed by standard inter-ECU messages. For example, there is no message that will let another ECU completely disable the brakes, and especially not for an extended period of time. In these instances, diagnostic services can often be abused to achieve the desired effect.

One powerful diagnostic service is the ability to override device I/O. While the exact payload of these message varies by OEM and ECU, the UW/UCSD team found it extremely easy to enumerate virtually every possible behavior by just sending random payloads. Combined with elevated privileges obtained by exploiting weak SecurityAccess schemes, an attacker can potentially perform dangerous operations, such as taking direct control of the brakes while the vehicle is moving at high speed.

Another useful diagnostic service is ReadMemoryByAddress, which can enable an attacker to read arbitrary pieces of an ECU's address space. This service can often be used to dump an ECU's firmware for reverse-engineering or to leak sensitive values such as authentication keys. While suppliers are cautioned to prevent leaking sensitive data over this service, many do not heed this warning. Others may not implement ReadMemoryByAddress restrictions correctly. For example, an ECU may prevent you from reading out sensitive values from flash, but does not prevent you from reading the same values out when they are copied to RAM.

Finally, the RequestDownload/TransferData services can be used to reflash ECUs, which allows an attacker to implement arbitrary behavior. These services should normally be restricted, but in many cases they aren't, and in other cases the SecurityAccess mechanism protecting access can often be defeated.

Summary

Modern automobiles have dozens of control units that communicate with each other via CAN buses. CAN buses are a shared broadcast medium, and while they are designed for reliability, they aren't designed to withstand malicious attacks. Many critical aspects of a vehicle's operation can be controlled with access to these buses, either by spoofing ordinary inter-ECU messages or by abusing diagnostic services. These CAN buses are becoming increasingly vulnerable to attack. Aftermarket devices plugged into the ODB-II port are in a position of privileged access and may be vulnerable to wireless attacks. Furthermore, vehicles themselves are now incorporating wireless connectivity (e.g., Bluetooth, WiFi, and cellular) in their infotainment and telematics systems, further broadening the potential attack surface. However, with recent media attention on these types of vulnerabilities, we are hopeful that automakers and aftermarket device manufacturers will devote more resources to securing their products.

References

[1] U. Kiencke, S. Dais, and M. Litschel, "Automotive Serial Controller Area Network," SAE Technical Paper 860391, 1986: doi:10.4271/860391.

[2] International Organization for Standardization, "Road Vehicles— Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication," ISO/DIS Standard 11898:1993.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, "Experimental Security Analysis of a Modern Automobile," in the *Proceedings of the 31st IEEE Symposium on Security and Privacy*, May 16–19, 2010, Oakland, CA.

[4] C. Miller and C. Valasek, "A Survey of Remote Automotive Attack Surfaces," Black Hat USA 2014, August 2014, Las Vegas, NV.

[5] International Organization for Standardization, "Road Vehicles—Diagnostics on Controller Area Networks (CAN)—Part 2: Network Layer Services," ISO Standard 15765-2:2004.

[6] C. Miller and C. Valasek, "Adventures in Automotive Networks and Control Units," DEF CON 21, July 2013, Las Vegas, NV.

[7] International Organization for Standardization, "Road Vehicles—Diagnostics on Controller Area Networks (CAN)—Requirements for Emissions-Related Systems," ISO Standard 15765-4:2005(E).

[8] I. Foster, A. Prudhomme, K. Koscher, S. Savage, "Fast and Vulnerable: A Story of Telematic Failures," in the *Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT '15)*, August 2015, Washington, DC.

[9] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in the *Proceedings of the 20th USENIX Security Symposium*, August 2011, San Francisco, CA.

[10] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," Black Hat USA 2015, August 2015, Las Vegas, NV.

ENIGMA

It's time for the security community to take a step back and get a fresh perspective on threat assessment and attacks. This is why the USENIX Association is excited to announce the launch of Enigma, a new security conference geared towards those working in both industry and research.

Enigma will deliver three days of talks from leading practitioners and researchers, all of whom are uniquely qualified to discuss security as it relates to the Internet of Things, black markets, election issues, threats, scalability, and much more.

Featured speakers include:

Bryan Payne, Netflix:
"PKI at Scale Using Short-Lived Certificates"

Adrienne Porter Felt, Google:
"Why Is Usable Security Hard, and What Should We Do about It?"

Damon McCoy, New York University:
"Bullet-Proof Credit Card Processing"

The full program and registration are now available.

enigma.usenix.org

JANUARY 25–27, 2016

SAN FRANCISCO, CALIFORNIA, USA

usenix ASSOCIATION

The Expanding World of Digital Forensics

SIMSON L. GARFINKEL

Simson Garfinkel was the program chair of the DFRWS 2015 Conference and researches digital forensics in Arlington VA. He is also an Affiliate Faculty Member of George Mason University in Virginia. simsong@acm.org

Digital forensics is on its way to becoming a mainstream part of computer science. With the right tools, forensic examiners can trace the source and determine the extent of a cyberattack. They can find evidence on a cell phone to help convict a bank robber or pimp. They can pull apart the firmware in cars and embedded devices, finding privacy leaks and security vulnerabilities. To accomplish this wizardry, the field has had to grow beyond its roots of simple data extraction and file system analysis, and now incorporates a wide variety of leading-edge computer science techniques, including big data analytics, visualization, multilingual processing, and program analysis. It's never been a better time to be a digital forensics researcher—the people who are charged with discovering how to exploit new technologies and building the tools that systemize that knowledge. At the same time, increased technical complexity and diversity is making the job of front-line forensics examiners more challenging every day.

Digital forensics is fragmenting. Ten years ago, it was common for examiners to be masters of the entire field—and perhaps developing their own tools as well. But as computer systems have become more complex, there's been an increasing need for examiners to specialize. As a result, having one or a few forensic specialists on staff is no guarantee that an organization can perform the necessary forensic tasks when the times arise. Instead, organizations increasingly rely on specialized teams that deeply practice and research a particular modality, then use partnerships to cover other forensic areas.

This article provides a brief overview of digital forensics as it is practiced today. I then present some of the recent advances in digital forensics research. Finally, I discuss the profound changes the field is likely to encounter over the next few years as a result of the growing attention that society is paying to privacy and security.

Digital Forensics Comes of Age

Modern digital forensics got going in the 1990s when law enforcement agents started encountering digital media during the course of criminal investigations. Some of these were classic examples of what we call cybercrime today—an outsider breaking into some kind of networked computer system, or an authorized user planting malware. Other cases were traditional crimes involving drugs, theft, or extortion, with the added twist that a computer system was used by the suspect to convey a threat, keep records, or communicate with co-conspirators.

The Internet's explosive growth in the late 1990s was accompanied by a similar increase in child pornography proliferation [1]. At the time, much of the public discussion focused on technologies for detecting and preventing criminals from downloading child porn over the network. But when law enforcement actually made an arrest, agents faced computers that needed to be examined. As the cases started to mount, agents at the FBI and other law enforcement agencies realized that they needed standardized, repeatable, and accepted approaches for preserving digital evidence, making digital duplicates for use by examiners, searching for items of interest, and other aspects of the new digital laboratory practice [2].

To understand why standards were necessary, consider the example of deleted file recovery. Since the 1980s, programs like Norton Disk Doctor and Mace Utilities could "unformat" hard drives and "undelete" files that users accidentally deleted [3]. Law enforcement professionals were interested in such tools as well, since perpetrators would frequently delete files or otherwise try to hide their criminal activity if they suspected they were in danger of being apprehended. Evidence extracted from a suspect's own system proved to be incredibly useful for establishing guilt.

But it's one thing for a consumer to use an off-the-shelf program to recover a file that's been accidentally lost, another thing entirely for a law enforcement officer to recover a file on a suspect's computer and submit that file as evidence in a court of law. In the latter case, the officer needs some way to prove that the "deleted" file was really on the suspect's computer and not the result of contamination from another case or from running the recovery tool. To address these and similar concerns, practitioners developed techniques for both preserving and isolating case data. Eventually, the Federal Crime Laboratory Directors formed the Scientific Working Group on Digital Evidence (SWGDE) in February 1998 to help formalize the profession's standards [4].

Vendors responded by perfecting software that would reliably copy all of the data from a hard drive into a single "image file" (complete with checksums, case notes, and timestamps), and devices called "write blockers" that fit between a hard drive and a computer, allowing data to be copied off the drive but blocking attempts from the computer to overwrite sectors on the drive. Disk imaging proved to be more complex than first thought, as data could be hidden (and occasionally was hidden) in the "host protected area" or "device configuration overlay" of ATA-hard drives. Likewise, some hard drives contain bad blocks, and many imaging tools did not behave in a reliable and consistent manner when bad blocks were encountered [5]. Hardware write blocking proved necessary because many operating systems would overwrite sectors on a hard drive even when the drive was mounted "read-only." For example, some Linux "live CD" distributions will mount and use a Linux swap partition, potentially overwriting important evidence [6]. Write blockers also protected against an examiner's mistakes.

The aftermath of September 11, 2001, demonstrated that the power of digital forensics to recover deleted documents and report about a computer's past usage could be used for more than child exploitation cases. Although the hijackers are reported to have had no computers of their own, investigators searched computers in public libraries and copy shops frequented by the hijackers and discovered the systems had been used "to review and order airline tickets" used in the attack [7]. Computer records, including data found on the laptop of Zacarias Moussaoui, the alleged "20th hijacker," were featured prominently at Moussaoui's trial and significantly assisted the prosecution [8]. In the years that followed, digital forensics was widely used by coalition forces in Iraq and Afghanistan to gain intelligence from captured cell phones and laptops [9]. In May 2015, for example, the Office of the Director of National Intelligence released a trove of documents that had been seized as part of the 2011 raid on Osama bin Laden's compound [10].

Data extraction and file recovery remained primary goals of digital forensics researchers and developers in the decade following the 9/11 attacks. Smartphones increased the importance of digital forensics. Far more personal than a laptop, smartphones are often intimately involved in the planning and commission of crimes. Criminals use smartphones to communicate with their co-conspirators and with their victims (in the case of sexual assault) and even to document their crimes [11]. Unfortunately, it can be quite complicated to get the information out of a smartphone—unlike a hard drive, an examiner can't simply connect the phone to a "write blocker" and copy out all of the data. But once a phone's memory is dumped, the use of SQLite databases, JSON data structures, and text files made phone content relatively straightforward for an examiner to understand.

Many forensic processes designed for copying and analyzing data from simple IDE hard drives have been adopted for RAID arrays, SSDs, digital cameras, GPS devices, mobile phones, and an increasingly dizzying array of devices. There have been challenges. For example, many modern systems do not statically preserve data in the same way that magnetic drives do—SSDs that implement the TRIM command will slowly clear the blocks associated with deleted files if the drive is powered up, without any help from the operating system. This requires changing not just operating procedures, but underlying assumptions about the nature and goals of digital forensics.

Today there are mature commercial and open source tools available for digital forensics practitioners. Programs like EnCase, FTK, X-Ways Forensics, and Autopsy allow an examiner to view the contents of a disk image, perform keyword searches, and even recover deleted files. These tools typically support a range of file systems, including Windows FAT, XFAT and NTFS, Macintosh HFS+, and Linux EXT 2/3/4. A similar set of tools from companies like Celebrite and NowSecure provide these functions for iOS and Android-based phones. These tools implement a model I call "visibility, filter and report." First, they find all of the forensically interesting data on the media being analyzed and make them visible, typically by putting them in a database and displaying a section of that database in a graphical user interface. Next, they allow the examiner to filter the data according to rules. Finally, the tools produce reports such as timelines, indications of known malware, and user activities. These tools have proven to be remarkably effective in helping examiners perform a wide variety of functions, from malware analysis to crime fighting.

Handling of non-English text has become increasingly important as examiners routinely encounter text in other languages. Early forensic tools simply could not display many non-US languages. Today, support for UNICODE is uneven but improving, and some tools are beginning to incorporate machine translation, allowing examiners to get the sense of a case without having to bring in a human translator.

Forensics has also grown beyond analyzing data at rest. Consider network forensics. Fifteen years ago, civil libertarians were outraged over the FBI's development and use of a network wiretapping program called "Carnivore" [12]. But at roughly the same time, network examiners were developing an open source network forensics program called Ethereal, now known as Wireshark. Unlike Carnivore, Wireshark can decrypt SSL-encrypted traffic (provided that the examiner has a copy of the server's private key). It's a great tool for "forensicating" networks, as the pros call it. Meanwhile, there's a whole new generation of malware analysts who consider digital forensics to be the study of opcodes, execution paths, and trust elevation exploits.

Digital Forensics Research

"Digital forensics" has become an umbrella term for any systematic examination of digital artifacts, code and data, no matter where they may be formed. And as the field has grown and matured, so too has the need for systematic research into the processes that create those digital artifacts and techniques for helping examiners to make sense of the massive amount of information that systematic examination produces.

The idea of digital forensics as an area of academic or industrial research dates to August 2001, when the Air Force Research Laboratory sponsored a two-day workshop in Utica, New York, on Digital Forensics Research. The results of that workshop were a 42-page report, "A Roadmap for Digital Forensics Research," and the annual Digital Forensics Research Workshop, later renamed "DFRWS."

Much of the past 15 years of research has been devoted to understanding the nature of stored data. In practice, there is little publicly available documentation for the vast majority of systems that have been deployed. An added complication is that many vendors have made their own changes to the data structures used by various operating systems and applications, sometimes in an attempt to get better performance from these systems, other times because they were not interested in maintaining compatibility with their competitors' systems. But even when software is open source, there is a big difference between having a copy of a program's source code and being able to understand the information that a program writes into a file, in a database, or on a disk.

Consider the case of SQLite, the cross-platform open source database that's widely used on mobile phones and desktop computers. Even though SQLite's code has been public since its initial release in May 2000, it's only in the past few years that forensic examiners have understood how to recover data that's been deleted in SQLite databases or partially overwritten database files. The reason: simply having a program's code may give insight into how the programs run, but the only way to really understand the data that a complex program produces is to methodically trace the program's execution and painstakingly track the output. In hindsight, this is just another application of the Church-Turing thesis.

Likewise, even though Microsoft has published specifications that describe FAT file systems in great detail and there have been open source implementations for nearly two decades, neither explains how to recover deleted files, or how to carve FAT32 directories from a drive that was reformatted by a computer running Windows XP, a version of Windows for which formatting wiped the root directory of the drive but did not overwrite most of the drive's actual storage.

Researchers have also spent considerable effort understanding the internal structure of various file formats. Simply trying to identify a file's "type" and extract the file's "text" proved to be a difficult problem in many cases. Although identifying a Microsoft Word document is relatively easy, there are thousands of different word processors, graphics tools, and financial programs in use. In many cases different versions of these programs write files with subtly different formatting. Since it's frequently not practical for examiners to acquire and run the precise version of the software that was used to create a file, other approaches need to be developed for understanding formats and extracting their text. Understanding these formats has another benefit as well: frequently, files contain hidden information that the normal end-user application doesn't show. Such information can be used to gain additional insight or intelligence about a crime or criminal organization.

Some of the most interesting data analysis work involves the reconstruction of files when some information is overwritten or missing but other binary data remains. For example, file carving is an approach for extracting files from media based on their content, rather than using file system metadata. Early file carvers could find and identify JPEGs by searching for the two characteristic bytes that start and end every JPEG file (hex FF D8 and FF D9, respectively), then saving the header, the footer, and all of the bytes between into a new file. This approach, called header-footer file carving, generated a large number of false positives. Researchers discovered ways for discarding the invalid JPEGs. Researchers have since developed techniques for reassembling fragmented JPEG files [13], and even for rendering a fragment of a JPEG photograph when large parts from the file's beginning and end are missing and all that's available are the blocks in the middle [14]. These techniques need to be extended to the multitude of video formats.

Another important research area is memory analysis. Tools like the open source Volatility memory analysis framework [15] provide building blocks for understanding memory dumps, converting virtual memory addresses to physical, and decoding many of the operating system structures. Plugins written for Volatility can reconstruct the process list and find rootkits residing on Windows, Macintosh, and Linux systems. The contents of the Windows clipboard can be extracted and printed, open files can be displayed, and typed command lines can be found and recovered. Such tools have been tremendously important for incident response. Unfortunately, they are incredibly expensive to develop, creating the need for new development approaches.

Malware analysis has also become hugely important—so much so that it is largely its own area with specialty tools like Ida Pro and OllyDbg. Most malware analysis is done manually, with tools performing disassembly, search, and clerical support, but there is growing work in techniques that are largely automatic, relying either on static analysis or else running malware in environments that are highly instrumented.

Data extraction is still an important research area, although these days the hard problems are overcoming encryption and dealing with the data volumes of modern storage systems. For example, at the 15th DFRWS, which concluded August 2015 in Philadelphia, the Best Paper Award went to a pair of researchers who developed a technique for selectively imaging a small portion of a hard drive while still acquiring information necessary to solve a case [16]. Demonstrating the increased emphasis on scientific method and validation in the world of forensics, the paper characterized the accuracy of the technique with a synthetic publicly available data set [17] and on an actual case involving employee misconduct. Such techniques are soon to be extended to mobile devices and the cloud.

Privacy Cuts Both Ways

The public's increasing concern about digital privacy is increasingly a strong motivator and a growing barrier to digital forensics. A significant amount of research is now the result of work of researchers in related fields using current digital forensics tools and developing new approaches to perform privacy assessments of computers, mobile devices, and embedded systems. What they do with the results of these assessments will determine the future of the field.

For example, at the October 2015 ACM Conference on Computer and Communications Security, a pair of papers from researchers at Purdue explore sensitive information left in the memory of Android mobile phones. In one paper, the authors show that they can recover images that were taken with the camera but never stored in the phone's flash memory, such as frames from a Skype call or preview images from the camera [18]. In the second paper, the same authors show that they can recover Android GUIs from memory fragments that haven't been cleared [19]. These two papers rely on existing forensic techniques to image the phones' memory and provide new techniques that might be hugely useful in criminal investigations. On the other hand, these papers, and others in the same vein, provide developers with roadmaps of privacy leaks that need fixing—and in so fixing them, removing the possibility that the leaks might be used in future forensic examinations. In the past, forensics researchers typically did not widely publicize their findings for fear that vendors would fix the very privacy bugs that were helping to put criminals in prison.

Tool developers are actively searching for approaches that will let a forensics examiner visualize the massive amounts of data that a typical examination can recover. Examiners need tools that can automatically construct activity timelines, digest documents, and summarize video. These approaches need to automatically adjust themselves as the data scales multiply by orders of magnitude—from a few hundred photos that might be on a person's cell phone, to a few million that might reside on a server in a datacenter.

To leverage the attention of human examiners, some of the most important work being done on the algorithmic front is to identify new similarity techniques. The idea is to have digital forensics tools reliably find and cluster documents, photographs, and movie clips that are similar, so that examiners can spend their time looking at objects that are different from what's been seen before. Once clustered, other techniques like random sampling and machine learning could be used to characterize the variety.

These tools that can digest huge amounts of data are beginning to raise the concerns of civil liberties activists—just like in the days of Carnivore—who say that forensic capability needs to be weighed against privacy concerns. As a result, some jurisdictions are actively limiting the extent that examiners are allowed to search on a suspect's computer.

Meanwhile, forensic examiners are faced with the growing proliferation of devices, operating systems, and data formats. Examiners frequently master one kind of device just as something new shows up on the market. That keeps the job interesting, but it means that the tools and technologies used by the examiners are almost never current, and developers are constantly struggling to keep up. In the fast-changing world of digital forensics, what's needed most are not ways for automating forensic analysis, but faster ways for doing digital forensics research and deploying new tools.

References

[1] Kathryn C. Seigfried-Spellar, Gary R. Bertoline, and Marcus K. Rogers, "Internet Child Pornography, U.S. Sentencing Guidelines, and the Role of Internet Service Providers," in Gladyshev and Rogers (eds), *Digital Forensics and Cyber Crime: Third International ICST Conference*, ICDF2C 2011, Dublin, Ireland, October 2011.

[2] Carrie Morgan Whitcomb, "An Historical Perspective of Digital Evidence: A Forensic Scientist's View," *International Journal of Digital Evidence*, vol. 1, no. 1 (Spring 2002): https://utica.edu/academic/institutes/ecii/publications/articles/9C4E695B-0B78-10593432402909E27BB4.pdf.

[3] Peter McWilliams, "Mace Utilities Can Recover Disk Data That Drives Away," *Chicago Tribune*, February 1, 1987: http://articles.chicagotribune.com/1987-02-01/business/8701080849_1_hard-diskfiles-unformat.

[4] Scientific Working Group on Digital Evidence: https://www.swgde.org/.

[5] James R. Lyle and Mark Wozar, "Issues with Imaging Drives Containing Faulty Sectors," *Digital Investigation,* vol. 4 (September 2007), pp. 13–15: doi:10.1016/j.diin.2007.06.002.

[6] Ahmed Fathy Abdul Latif Mohamed, Andrew Marrington, Farkhund Iqbal, Ibrahim Baggili, "Testing the Forensic Soundess of Forensic Examination Environments on Bootable Media," *Digital Investigation*, vol. 11 (2014), S22–29: http://www.dfrws.org/2014/proceedings/DFRWS2014-3.pdf.

[7] "9/11 Hijackers Used Public Libraries," *The Washington Times*, April 28, 2005: http://www.washingtontimes.com/news/2005/apr/28/20050428-115527-9817r/.

[8] Copies of those computer records, including a chilling letter from M. Atta requesting information about pilot training in the United States, can be found at http://www.vaed.uscourts.gov/notablecases/moussaoui/exhibits/prosecution.html.

[9] Stephen Pearson and Richard Watson, *Digital Triage Forensics: Processing the Digital Crime Scene*, Syngress Publishing, 2010.

[10] Documents that had been seized as part of the 2011 raid on Osama bin Laden's compound: http://www.dni.gov/index.php/resources/bin-laden-bookshelf.

[11] Dana Hedgpeth, "'Fire Selfies' after a Jealous Rage Lead to Maryland Man's Arrest," *The Washington Post*, September 1, 2015: https://goo.gl/nOq9x6.

[12] For a full description of the Carnivore program, including more than a thousand pages of documents obtained under the Freedom of Information Act, see https://epic.org/privacy/carnivore/.

[13] Anandabrata Pal, Husrev T. Sencar, Nasir Memon, "Detecting File Fragmentation Point Using Sequential Hypothesis Testing," *Digital Investigation*, vol. 5 (2008), S2–S13.

[14] Jusrev T. Sencar and Nasir Memon, "Identification and Recovery of JPEG Files with Missing Fragments," *Digital Investigation*, vol. 6 (2009), S88–S98.

[15] The Volatility Foundation, Volatility Memory Forensics Framework: http://www.volatilityfoundation.org/.

[16] Jonathan Grier and Golden G. Richard III, "Rapid Forensic Imaging of Large Disks with Sifting Collectors," *Digital Investigation*, vol. 14 (2015), S34–44.

[17] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing Science to Digital Forensics with Standardized Forensic Corpora," *Digital Investigation*, vol. 6 (2009), S2–11.

[18] Brendan Saltaformaggio, Rohit Bhatia, Zhongshu Gu, Xiangyu Zhang, Dongyan Xu, "VCR: AppAgnostic Recovery of Photographic Evidence from Android Device Memory Images," ACM CCS, October 2015.

[19] Brendan Saltaformaggio, Rohit Bhatia, Zhongshu Gu, Xiangyu Zhang, Dongyan Xu, "GUITAR: Piecing Together Android App GUIs from Memory Images," ACM CCS, October 2015.

Cryptanalysis of the Megamos Crypto Automotive Immobilizer

ROEL VERDULT AND FLAVIO D. GARCIA

Roel Verdult performed scientific research in a variety of security topics, including electronic passports, contactless smart cards, radio frequency identification (RFID), near field communication (NFC), secure storage, authentication protocols, and other types of transmission security. The relevance of his work is demonstrated by his numerous significant international research awards. He earned his doctorate at two universities, receiving a dual degree from Radboud University, the Netherlands, and KU Leuven, Belgium. Currently, Roel Verdult is active as a cryptographic research engineer and is co-founder of the Dutch IT Security & Engineering company FactorIT BV. roel@factorit.nl

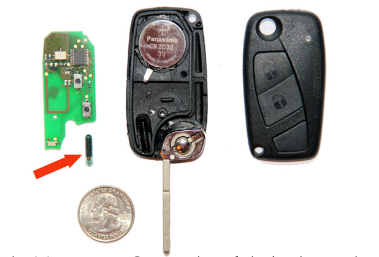
Dr. Flavio Garcia is a Senior Lecturer (Associate Professor) and Senior Birmingham Fellow at the University of Birmingham in the UK. His work focuses on the design and evaluation of cryptographic primitives and protocols for embedded devices like automotive key fobs and smart cards. His research achievements include breakthroughs such as the discovery of vulnerabilities in the four most widely used contactless smart cards: the Mifare Classic, HID iClass, and Atmel's SecureMemory and CryptoRF. The first of these, Mifare Classic, was widely used for electronic payment (e.g., Oyster Card) and access control (e.g., Amsterdam Airport). f.garcia@bham.ac.uk

The Megamos Crypto key fob is used in one of the most widely deployed automotive electronic immobilizers. Such an anti-theft device is designed to prevent hot-wiring of the vehicle. We have reverse-engineered all proprietary security mechanisms of the key fob and have found several weaknesses in the cipher and also in their usage and configuration by carmakers. We exploit these weaknesses in three practical attacks that recover the 96-bit key fob secret key. We end our article with suggestions to mitigate some of our attacks, something that knowledgeable users can do themselves.

Electronic vehicle immobilizers have been very effective at reducing car theft. Such an immobilizer is an electronic device that prevents the engine of the vehicle from starting when the corresponding key fob is not present. This key fob is a low-frequency RFID chip typically embedded in the vehicle's key. When the driver starts the vehicle, the car authenticates the key fob before starting the engine, thus preventing hot-wiring. In newer vehicles the mechanical ignition key has often been removed and replaced by a start button (see Figure 1a). In such vehicles the immobilizer key fob is the only anti-theft mechanism that prevents a hijacker from driving away with the vehicle. In some countries, having such an immobilizer is enforced by law. For example, according to European Commission directive (95/56/EC) it is mandatory that all cars sold in the EU from 1995 on be fitted with an electronic immobilizer. Similar regulations apply to other countries like Australia, New Zealand (AS/NZS 4601:1999), and Canada (CAN/ULC S338-98). Although it is not required by law in the US, according to the independent organization Insurance Institute for Highway Safety (IIHS), 86 percent of all new passenger cars sold in the US had an engine immobilizer installed.

An electronic car immobilizer consists of three main components: a small key fob chip embedded in (the plastic part of) the car key (Figure 1b); an antenna coil located in the dashboard of the vehicle, typically around the ignition barrel; and the immobilizer unit that prevents the vehicle from starting the engine when the key fob is absent.

1a: Keyless ignition with start button



1b: Megamos Crypto key fob (indicated by arrow) in a car key

Figure 1: Megamos Crypto integration in vehicular systems

Make	Models
Alfa Romeo	147, 156, GT
Audi	A1, A2, A3, **A4 (2000)**, A6, **A8 (1998)**, Allroad, Cabrio, Coupé, Q7, S2, S3, S4, S6, S8, **TT (2000)**
Buick	Regal
Cadillac	CTS-V, SRX
Chevrolet	Aveo, Kalos, Matiz, Nubira, Spark, Evanda, Tacuma
Citroën	**Jumper (2008)**, Relay
Daewoo	Kalos, Lanos, Leganza, Matiz, Nubira, Tacuma
DAF	CF, LF, XF
Ferrari	California, 612 Schaglietti
Fiat	Albea, Doblo, Idea, Mille, Multipla, Palio, **Punto (2002)**, Seicento, Siena, **Stilo (2001), Ducato (2004)**
Holden	Barina, Frontera
Honda	Accord, Civic, CR-V, FR-V, HR-V, Insight, **Jazz (2002, 2006)**, Legend, Logo, S2000, Shuttle, Stream
Isuzu	Rodeo
Iveco	Eurocargo, Daily
Kia	Carnival, Clarus, Pride, Shuma, Sportage
Lancia	Lybra, Musa, Thesis, Ypsilon
Maserati	Quattroporte
Opel	Frontera
Pontiac	G3
Porsche	911, 968, Boxster
Seat	Altea, Cordoba, **Ibiza (2014)**, Leon, Toledo
Skoda	**Fabia (2011)**, Felicia, Octavia, Roomster, Super, Yeti
Ssangyong	Korando, Musso, Rexton
Tagaz	Road Partner
Volkswagen	Amarok, Beetle, Bora, Caddy, Crafter, Cross Golf, Dasher, Eos, Fox, Gol, **Golf (2006, 2008)**, Individual, Jetta, Multivan, New Beetle, Parati, Polo, Quantum, Rabbit, Saveiro, Santana, **Scirocco (2011)**, Touran, **Tiguan (2010)**, Voyage, **Passat (1998, 2005)**, Transporter
Volvo	C30, **S40 (2005)**, S60, S80, **V50 (2005)**, V70, XC70, XC90, XC94

Table 1: Vehicles that used Megamos Crypto for some version/year. Boldface and year indicate specific vehicles we experimented with.

The immobilizer unit communicates through the antenna coil and enumerates all key fobs that are in proximity of the field. The key fob identifies itself and waits for further instructions. The immobilizer challenges the key fob and authenticates itself first. On a successful authentication of the immobilizer unit, the key fob sends back its own cryptographic response, which is different every time. Only when this response is correct does the immobilizer unit enable the engine to start.

The immobilizer unit is directly connected to the internal board computer of the car, also referred to as the electronic control unit (ECU). To prevent hot-wiring a car, the ECU blocks fuel-injection, disables spark plugs, and deactivates the ignition circuit if the key fob fails to authenticate.

A distinction needs to be made between the vehicle immobilizer and the remotely operated central locking system. The latter is battery powered, operates at ultra-high frequency (UHF), and only activates when the user pushes a button on the remote to (un)lock the doors of the vehicle. Figure 1b shows a disassembled car key where it is possible to see the passive Megamos Crypto key fob and also the battery powered remote of the central locking system.

The Megamos Crypto key fob is the first cryptographic immobilizer key fob manufactured by EM Microelectronic-Marin SA and is currently one of the most widely used. The manufacturer claims to have sold more than 100 million immobilizer chips, including Megamos Crypto key fobs [4]. Table 1 shows a list of vehicles that use or have used Megamos Crypto at least for some version/year. As can be seen from this list, many Audi, Fiat, Honda, Volkswagen, and Volvo cars used Megamos Crypto key fobs.

The key fob uses a 96-bit secret key and a proprietary cipher in order to authenticate to the vehicle. Furthermore, a 32-bit pin code is needed in order to be able to write on the memory of the key fob. The concrete details regarding the cipher design and authentication protocol are kept secret by the manufacturer, and little is currently known about them.

From our collaboration with the local police it was made clear to us that sometimes cars are being stolen and nobody can explain how. They strongly suspect the use of so-called “car diagnostic” devices. Such a device uses all kinds of custom and proprietary techniques to bypass the immobilizer and start a car without a genuine key. This motivated us to evaluate the security of vehicle immobilizer key fobs.

In the last decades, semiconductor companies introduced several proprietary algorithms specifically for immobilizer security. Their security often depends on the secrecy of the algorithm, contrary to Kerckhoffs’ principle. When their inner-workings are uncovered, it is often only a matter of weeks before the first attack is published. There are several examples in the literature that address the insecurity of proprietary algorithms [5]. There are four widely used immobilizer key fobs that depend on proprietary cryptography: DST, KeeLoq, Hitag2, and Megamos Crypto, which were all proven to be insecure [1, 2, 7, 8]. The Megamos paper was accepted at the 22nd USENIX Security Symposium, but appears as an addendum to the 24th’s Proceedings, and is used as a basis for this article.

Hardware Setup

We used a Proxmark III (http://www.proxmark.org/) to eavesdrop and communicate with the car and key fob. This is a generic RFID protocol analysis tool that supports raw data sampling of radio frequency signals [6]. We have developed a generic open

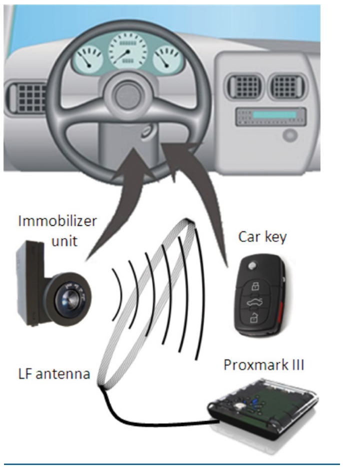


Figure 2: Experimental setup for eavesdropping

source library, which is capable of supporting any custom and proprietary RFID communication scheme that operates at a frequency of 125 kHz. This allowed us to implement a custom firmware and FPGA design that uses the modulation and encoding schemes of Megamos Crypto key fobs.

Furthermore, we added RFID reader/programmer functionality to send simple commands like read and write to the key fob. In particular, this library can be used to set the memory lock bit and a random pin code as a mitigation for our second attack, as described later in this article. Finally, we implemented an advanced firmware, which contains all cryptographic operations and is fully compatible with the Megamos Crypto authentication protocol. This enabled us to perform practical experiments with cars by eavesdropping and emulation of Megamos Crypto key fobs. However, we will not release any attack tools such as this advanced firmware.

Megamos Crypto

This section gives a short introduction to the workings of the Megamos Crypto key fob. It briefly introduces the cryptographic algorithms and protocols used in Megamos Crypto; a more detailed description is available in [8].

Authentication Protocol

The car authenticates by sending a random nonce n_C and the corresponding car authenticator a_C. When the car successfully authenticates itself, the Megamos Crypto key fob responds with its own key fob authenticator a_T back to the car. A simplified version of the Megamos Crypto authentication protocol is shown in Figure 3.

When the driver turns on the ignition, several messages between the car and key fob are exchanged. It starts with the car reading out the key fob memory blocks. Next, the car tries to authenticate using the shared secret key k. If the authentication fails, the car retries around 20 times before it reports on the dashboard that the immobilizer failed to authenticate the key fob. Table 2 shows an eavesdropped trace of a German car that initializes and authenticates a Megamos Crypto key fob using the 96-bit key 000000000000010405050905. The structure of the secret key of the car suggests that it has an entropy of only 24 bits.

Cryptographic Algorithm

Several after-market diagnostic and locksmith tools such as the *Tmpro2, MiraClone, AVDI,* and *Tango Programmer* implement the Megamos Crypto cipher for key fob production and verification. None of these tools is able to recover the secret key of a key fob or perform any kind of cryptanalysis. However, the software package that comes with Tango Programmer implements all cryptographic operations of the key fob, including the Megamos Crypto cipher. We have analyzed the software thoroughly and extracted the algorithm from it. The Megamos Crypto cipher is a stream cipher that consists of five main components: a 23-bit Galois Linear Feedback Shift Register, a 13-bit Non-Linear Feedback Shift Register, and three 7-bit registers.

The stream cipher basically works as a pseudo-random generator that is seeded by the secret key k and the car nonce n_C. It then runs producing pseudo-random bit-strings a_C and a_T, which are used in the authentication protocol as proof of knowledge of the secret key (see Figure 4).

Cryptanalysis of Megamos Crypto

In our full paper [8], we have proposed a cryptanalysis that compromises *all* vehicles using Megamos Crypto. This cryptanalysis requires an adversary to eavesdrop two successful authentication traces between the car and the key fob to recover the 96-bit secret key. We would like to emphasize that in order to get these two traces, a perpetrator needs access to both the car and the original car key. Our cryptanalysis reduces the computational complexity from 2^{96} (a brute force attack) to 2^{56} encryptions.

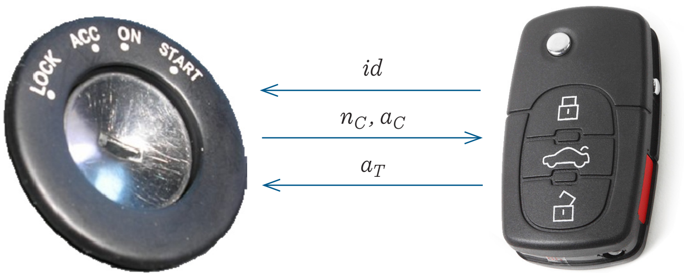


Figure 3: Megamos Crypto authentication protocol

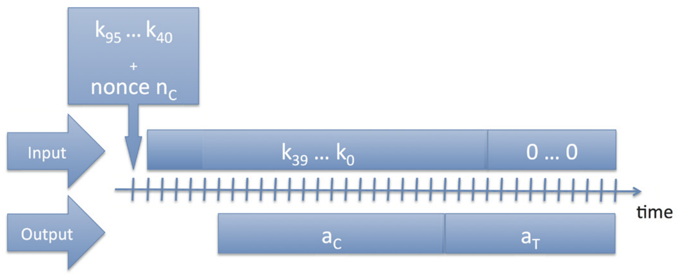


Figure 4: Initialization and propagation of the cipher

Origin	Message	Description
Car	3	Read identifier
Key fob	A9 08 4D EC	Identifier $id_{31} \dots id_0$
Car	6 | 3F FE 1F B6 CC 51 3F | 0^7 | F3 55 F1 A	Authentication, $n_{C_{55}} \dots n_{C_0}, 0^7, a_C$
Key fob	60 9D 6	Car authenticated successfully, send back a_T

Table 2: Eavesdropped Megamos Crypto authentication trace

This could be computed within two days on a COPACOBANA, which is an FPGA-based massively parallel computer.

Once the secret key is recovered, it is possible to emulate the original key fob, effectively cloning the original key. The cryptanalysis described above exploits the following weaknesses.

- The key fob lacks a pseudo-random number generator, which makes the authentication protocol vulnerable to replay attacks.
- The internal state of the cipher consists of only 56 bits, which is much smaller than the 96-bit secret key.
- The cipher state successor function can be inverted; given an internal state and the corresponding bit of cipher-text, it is possible to compute the predecessor state.
- The last steps of the authentication protocol provide an adversary with 15-bits of known-plaintext.

This computational complexity can be further reduced by a time/memory tradeoff. Many tradeoffs are possible, but if we were to use a 12-terabyte lookup table, for example, then the complexity is reduced to 2^{49} table lookups. This optimized version of the attack takes advantage of the fact that some of the cipher components can be run quite autonomously. Such a time-memory tradeoff, however, requires many indirect memory lookups and is therefore difficult to mount in practice with ordinary consumer hardware.

Partial Key-Update Attack

Currently, the memory of many Megamos Crypto key fobs in the field is either unlocked or locked with a publicly known default pin code. This means that anybody has write access to the memory of the key fob. This also holds for the secret key bits that make it vulnerable to a trivial denial of service attack. An adversary just needs to flip one bit of the secret key of the key fob to disable it.

Besides this obvious weakness, there is another weakness regarding the way in which the 96-bit secret key is written to the key fob. These 96 bits are stored in six memory blocks of 16 bits each. But it is only possible to write one block at a time to the key fob, which constitutes a serious weakness since a secure key-update must be an atomic operation.

This weakness enables an adversary to use a guess-and-determine technique in which she overwrites one block of the key at a time until she finds the complete secret key. For this attack we assume that an adversary is able to communicate with the car and key fob. A slightly optimized version of this attack requires only one successful authentication trace. In total, we need to write three times on the memory of the key fob and perform 3×2^{16} authentications with the key fob. This can be done within 30 minutes using a Proxmark III. The computational complexity of the last three steps is 2^{15} encryptions, which takes less than a second on an ordinary laptop.

We have executed this attack in practice and recovered the secret key of several cars from various makes and models. Having recovered the key, we were able to emulate the key fob and start the vehicles.

Weak-Key Attack

Our third attack is based on the following observation: many of the keys that we have recovered using the previous attack had very low entropy and exhibited a well-defined pattern, i.e., the first 32 bits of the key were all zeros. In the remainder of this paper we call such a key *weak*. This attack consists of a time-memory tradeoff that exploits this weakness to recover the secret key, within a few minutes, from two authentication traces. This attack requires storage of a 1.5 TB rainbow table.

Table 3 shows some examples of weak keys we found during our experiments (on the vehicles indicated in Table 1). To avoid naming concrete car models we use *A, B, C*...to represent car makes. We write numbers *X*.1, *X*.2, *X*.3...to represent different car models of make *X*.

Car	Secret key
A.1	00000000d8 b3967c5a3c3b29
A.2	00000000d9 b79d7a5b3c3b28
B.1	0000000000 00010405050905

Table 3: Recovered keys from our own cars. Besides the evident 32 leading zero bits, every second nibble seems to encode a manufacturer-dependent value, which further reduces the entropy of the key.

Apparently, some car manufacturers have decided to use only 64 bits of the secret key, probably due to compatibility issues with legacy immobilizer systems. If a Megamos Crypto key fob uses such a weak key, it is possible to recover this key quickly, even when the memory of the key fob is locked with a pin code. Concretely, if the first 32 bits of the key are constant (e.g., zeros), this allows an adversary to pre-compute and sort on 47 contiguous output bits for each internal state. However, such a table, with 2^{56} entries, requires a huge amount of storage. Many time-memory tradeoff methods have been proposed in the literature. For example, a rainbow table shrinks the storage significantly, while requiring only a modest amount of computation for a lookup. Just to give an impression of the feasibility of this attack, if we were using a rainbow table of 1.5 TB, then the computational complexity required to perform this attack would only be 2^{37} encryptions, which can be computed within a few minutes on a standard laptop.

Practical Considerations and Mitigation

Our attacks require close-range wireless communication with both the immobilizer unit and the key fob. It is not hard to imagine real-life situations, like valet parking or car rental, where an adversary has access to both for a period of time. It is also possible to foresee a setup with two perpetrators, one interacting with the car and one wirelessly pickpocketing the car key from the victim's pocket.

As mitigating measures, car manufacturers should set uniformly generated secret keys and, for the devices which are not locked yet, set pin codes and writelock their memory after initialization. These obvious measures would prevent a denial of service attack, our partial key-update attack described earlier, and our weak-key attack in the previous section.

Car owners can protect their own vehicles against a denial of service and the partial key-update attack. These attacks only work if the adversary has write access to the memory of the key fob, which means that the lock-bit is set to zero. It is possible for a user to test for this property with any compatible RFID reader, like the Proxmark III, using our communication library. If the lock-bit is set to zero, then you should set it to one. It is possible to set this bit without knowing the secret key or the pin code. When dealing with the more recent version of the Megamos Crypto key fob (EM4170), users should also update the pin code to a random bit-string before locking the key fob.

On the positive side, our first (cryptographic) attack is more computationally intensive than the other attacks, which makes it important to take the aforementioned mitigating measures in order to prevent the more inexpensive attacks. Unfortunately, our first attack is also hard to mitigate when the adversary has access to the car and the key fob (e.g., valet parking or car rental). It seems infeasible to prevent an adversary from gathering two authentication traces. Furthermore, this attack exploits weaknesses in the core of the cipher's design (e.g., the size of the internal state). It would require a complete redesign of the cipher to fix these weaknesses. To that purpose, lightweight ciphers like Grain, Present, and KATAN have been proposed in the literature and could be considered as suitable replacements for Megamos Crypto. Also, immobilizer products implementing AES are currently available in the market.

Conclusions

The implications of the attacks presented in this paper are especially serious for those vehicles with keyless ignition. At some point the mechanical key was removed from the vehicle, but the cryptographic mechanisms were not strengthened to compensate. We want to emphasize that it is important for the automotive industry to migrate from weak proprietary ciphers like this to community-reviewed ciphers such as AES and use them according to the guidelines. For a few years already, there have been contactless smart cards on the market that implement AES and have a fairly good pseudo-random number generator. It is surprising that the automotive industry is reluctant to migrate to such key fobs considering the cost difference of a better chip (≤ 1 USD) in relation to the prices of high-end car models (≥ 50,000 USD). Since most car keys are actually fairly big, the key fob design does not really have to comply with the (legacy) constraints of minimal size.

Following the principle of responsible disclosure, we notified the manufacturer of our findings back in November 2012. Since then we have maintained an open communication channel with them. We understand that measures have been taken to prevent the weak-key and partial key-update attacks when the key fob was improperly configured.

Acknowledgments

The authors would like to thank Ross Anderson, Tom Chothia, Riccardo Focardi, Dorine Gebbink, Casey Henderson, Bart Jacobs, Sam King, Bas Kortmann, Karsten Nohl, Carolyn Pike, Jon Rowe, Mark Ryan, Graham Steel for their firm support. We are also thankful to Robert Carolina and Kenny Paterson for getting involved [3].

References

[1] A. Bogdanov, "Linear Slide Attacks on the Keeloq Block Cipher," in *Information Security and Cryptology* (2008), vol. 4990 of *Lecture Notes in Computer Science*, Springer, pp. 66–80.

[2] S. C. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo, "Security Analysis of a Cryptographically-Enabled RFID Device," in *Proceedings of the 14th USENIX Security Symposium (USENIX Security 2005)* (2005), USENIX Association, pp. 1–16.

[3] R. Carolina, and K. G. Paterson, "Megamos Crypto, Responsible Disclosure, and the Chilling Effect of Volkswagen Aktiengesellschaft vs. Garcia et al.": http://www.isg.rhul.ac.uk/~kp/ Carolina-Paterson-Megamos-comment-20130828.pdf.

[4] 125 kHz crypto read/write contactless identification device, EM4170, product datasheet, March 2002, EM Microelectronic-Marin SA.

[5] R. Verdult, "The (In)security of Proprietary Cryptography," PhD thesis, Radboud University, The Netherlands, and KU Leuven, Belgium, April 2015.

[6] R. Verdult, G. de Koning Gans, and F. D. Garcia, "A Toolbox for RFID Protocol Analysis," in *Proceedings of the 4th International EURASIP Workshop on RFID Technology (EURASIP RFID 2012)* (2012), IEEE Computer Society, pp. 27–34.

[7] R. Verdult, F. D. Garcia, , and J. Balasch, "Gone in 360 Seconds: Hijacking with Hitag2," in *Proceedings of the 21st USENIX Security Symposium (USENIX Security 2012)* (2012), USENIX Association, pp. 237–252.

[8] R. Verdult, F. D. Garcia, and B. Ege, "Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer," in *Supplement to the 22nd USENIX Security Symposium (USENIX Security 2013)* (2015), USENIX Association, pp. 703–718.

Ethical Behavior in Cyberspace Research

JOHN MURRAY

John Murray is a Program Director in the Computer Science Laboratory at SRI International, Silicon Valley, CA. His research interests include interactive human-machine technologies, collaborative intelligence, multi-player game systems, and cognitive engineering. Prior to joining SRI, Dr. Murray held executive and technical leadership positions at several international information systems firms. He holds advanced engineering degrees from Stanford, the University of Michigan, and Dublin Institute of Technology in Ireland. jxm@sri.com

Traditional principles of scientific ethics for studies involving people are primarily designed to address direct *human-centered* research. However, online research in cybersecurity is inherently *data-centered* in nature. Consequently, cyber-researchers often operate with limited awareness or oversight of the potential human risks and effects of their activities. Newly proposed changes in the US policies and regulations governing human studies are likely to have a significant impact on the online research community. Furthermore, given the inherently transnational nature of online studies, there is a pressing need for harmonizing ethics observance regulations and guidelines for security research on virtual worlds, social network systems, and other cyber-environments across multiple jurisdictions.

At the USENIX 2015 Security Symposium, a panel of academic and industry experts focused on cybersecurity research ethics. The discussions specifically centered upon the dilemmas facing those involved in academic publishing—editors, reviewers, etc.—when confronted by articles that discuss cybersecurity explorations, which may reveal potential or real exposures or vulnerabilities.

The underlying questions of research integrity concern the beliefs and justifications that system developers, investigators, and experimenters use as the basis for undertaking their explorations, what types of impacts are considered and when, what benefits vs. harms tradeoffs are made, and so on.

While the moral dilemmas of revealing system vulnerabilities in academic publications are indeed important, they generally come towards the end of a (potentially lengthy) research effort, well after other damage may already have been done. In reality, the actual ethical challenges should be considered early in the process, when the research team is designing their initial investigations.

For example, suppose that cyber-investigators are exploring aspects of real-time online censorship in various countries. Their strategy is to find ways to initiate download requests across national boundaries for various forms of potentially controversial material. In order to accomplish this, the researchers gain unauthorized access to some individual devices in a targeted jurisdiction, which they use as proxy platforms for issuing their exploratory requests.

However, in their zeal to deploy their probes, they neglect to consider the possible adverse effects that their study might have on the owners or operators of the compromised systems. Such individuals or groups may be put at risk vis-a-vis their own government authorities, as a result of the investigators' actions. A tech-savvy ethics review of the research plans would probably have drawn attention to the potential problems and ensured that appropriate safeguards were put in place.

Many academic institutions and larger corporate organizations have ethics oversight panels or institutional review boards (IRBs), which are responsible for monitoring the human health and safety of experimental subjects, and attending to the potential for exploitation or coercion, especially among vulnerable populations. The historical background for IRBs grew out of the revelation of numerous misguided and abusive scientific studies during the twentieth century. The result was the introduction of policies and standards based on the 1979 Belmont Report [1], which is still used today to guide scientific ethics reviews across the US and beyond. These guidelines translate into government regulations for several categories of human subject research, in particular those that are supported by US federal funding.

Although these guidelines and regulations have continued applicability to classical human research laboratory work in fields like medicine and psychology, they have limited practical relevance to modern behavioral studies that involve highly networked information and communications technology (ICT) systems. These impracticalities are exacerbated by the pervasive need to undertake comprehensive, transnational experimental projects, where much of the human data collection and analysis is undertaken remotely across varied, and often incompatible, legal regimes and social norms. Yet such is the case for numerous researchers nowadays, who are working not just in ubiquitous social networks and popular gaming worlds, but also with online educational environments, cybersecurity applications, and monitoring/surveillance systems.

In consideration of these challenges, a 2011 update to the earlier guidelines, called the Menlo Report [2], was specifically developed to address issues of online security, privacy, anonymity, and other personal identifiable information (PII) concerns. The report's authors recognized that the broad cyber-research community needs a more rational and coordinated strategy for managing ethics observance, which particularly considers the scope and needs of ICT research. Such a tailored approach should emphasize studies of human behavior and community activity online, and apply across multiple jurisdictions in interactive professional and social environments.

This transition of some of these concerns into formal policies and regulations recently moved forward with the publication of a Notice of Proposed Rulemaking (NPRM) in the US Federal Register [3]. This serves to promote conversation and comment from parties affected by the proposed changes. The latest comment period is open until December 2015, after which revisions to the proposal will be considered in light of comments received.

As they currently stand, some of the proposed changes may have significant implications for transnational cyber-research. One key concern is the extent that they might exacerbate the differences between human subjects research requirements in the US and elsewhere, while at the same time relaxing some of the more stringent requirements that currently apply to the US research community.

Traditional ethics reviewers try to ensure equitable distributions of burdens and benefits among the human subjects actually involved in the study. However, as noted in the example earlier, online research activity may adversely affect innocent bystanders and neutral nonparticipants. Given the risks associated with real-time data-intensive experiments, such studies might better be reviewed in terms of *human-harming* research rather than human subjects research.

For example, solid contingency and response plans are needed for mitigation of realized harms, especially for low-probability/high-impact events. These types of safety monitoring procedures are standard in traditional biomedical studies, but are rarely considered in ICT research. Furthermore, when research involves surveillance, profiling, or monitoring, additional vulnerability protections are needed to prevent the misuse of findings and results. This is particularly the case when novel mergers of partial data from several public sources may produce PII that is not individually available from just one of them. Other concerns arise from the potential for abuse of data for social discrimination, especially by non-investigators.

Provisions are required to ensure conformance with international regulations on transborder data flow that include personal information. In this regard, the current oversight policies and data handling processes for multi-jurisdictional ethics approvals are primarily centered upon the requirements of pharmaceutical drug trials, medical device tests, etc., rather than on the research needs in global-scale social science, human-machine systems, and ICT.

To address this gap, an international ethics observance organization is needed, which would coordinate/oversee regulations and guidelines for research in online systems and other cyber-environments across multiple jurisdictions. This could be a consortium of nonprofit organizations in several domains, which would ensure smooth transnational processing of approvals. It seems appropriate that such a consortium would need to have the backing of a recognized international entity such as UNESCO.

The first steps toward such harmonization could be merely a matter of coordinating and making available the critical features of each local research context, or it could extend to negotiating safe harbors for compliance with a local research context. Thus, if a study complies with certain key components, then it is deemed to satisfy local research context requirements for specific countries. Another, further step might be to aim for legislative harmonization on the topic of research protection.

The bottom line is that almost any form of standardized ethical framework would help cyberspace researchers worldwide become more aware of the challenges and know when they have addressed some required basic considerations. This must be better than the current haphazard obstacle course, which generally leaves everyone guessing as to what they still need to do to work through this ethical minefield.

References

[1] Belmont Report: www.hhs.gov/ohrp/humansubjects/guidance/belmont.html.

[2] Menlo Principles: www.dhs.gov/sites/default/files/publications/CSD-MenloPrinciplesCOMPANION-20120103-r731_0.pdf.

[3] Notice of Proposed Rulemaking, Protection of Human Subjects: www.federalregister.gov/articles/2015/09/08/2015-21756/federal-policy-for-the-protection-of-human-subjects.

Hack, Play, Win
Lessons Learned Running the Maryland Cyber Challenge

RICHARD FORNO

Dr. Richard Forno directs the University of Maryland Baltimore County's Graduate Cybersecurity Program, serves as the Assistant Director of UMBC's Center for Cybersecurity, and is a Junior Affiliate Scholar at the Stanford Law School's Center for Internet and Society (CIS). His 20-year career spans the government, military, and private sectors, including helping build a formal cybersecurity program for the US House of Representatives, serving as the first Chief Security Officer for the InterNIC, and co-founding the Maryland Cyber Challenge. Richard was also one of the early researchers on the subject of "information warfare," and he remains a longtime commentator on the influence of Internet technology upon society. rforno@umbc.edu

Cyber competitions are a popular way for cybersecurity practitioners to develop operational skills and acquire and demonstrate abilities and competence in a range of technical and non-technical knowledge areas in the quest for prizes and bragging rights. I describe how the lessons from current competitions can help future competition organizers run successful challenges of their own, and discuss whether such events are sufficient to prepare the next generation of cybersecurity professionals.

An oft-cited and prominent concern facing the Internet security community is the need to identify and hire qualified cybersecurity practitioners able to fill critical technical, analytical, and managerial positions within the global technology workforce. A 2014 report from the Education Advisory Board [1] discusses the "exploding" demand for qualified cybersecurity practitioners, noting that cybersecurity jobs grew by 73% between 2007 and 2012 compared to 6% in all other industry sectors. Similarly, Burning Glass Technologies, a national employment research firm, notes that there are nearly 23,000 available cybersecurity positions in the Washington, DC metropolitan area [2]. Nowhere is this need more evident, or discussed more frequently, than in Maryland, a region some dub the "epicenter of cybersecurity" education, research, and industry [3].

In response to this concern, events in the cybersecurity discipline known as "cyber competitions" or "cyber challenges" seek to motivate and encourage high school and college students toward careers in cybersecurity by developing their technical and teamwork skills while also allowing more experienced cybersecurity professionals an opportunity to practice their expertise in a challenging venue for professional recognition. As a form of intellectual competition, these events are becoming increasingly popular and widespread; industry security conferences like DEFCON CTF or the Department of Defense DC3 Digital Forensics Challenge, and competitions within educational communities such as the National Cyber League (NCL), CyberPatriot, or the Collegiate CyberDefense Competition (CCDC) are but a few examples of prominent cyber challenges drawing worldwide participation. Other competitions, both large and small, continually are under development, as is a National Science Foundation-backed effort to create a national federation [4] to support and standardize the rules, activities, and conduct of cyber competitions.

Given the popularity of these events, and the ongoing global desire to launch new ones, I will draw upon the experiences of organizing and coordinating the Maryland Cyber Challenge in offering advice to current and future cyber competition planners. While no event will ever run perfectly, organizers must always strive to "get it right"—or as close to "right" as possible!

Event Background

As one of the many cyber competitions emerging in recent years, the Maryland Cyber Challenge (MDC3) is a prominent regional and innovative approach to cybersecurity competitions in support of Maryland's declared leadership in cybersecurity education, research, and industry. However, unlike most cyber competitions, MDC3 is a multi-division event that simultaneously hosts competitors in high school, college, and professional categories—although teams only compete within their respective division and for separate, quite meaningful, prizes.

The challenge is organized around two virtual qualification rounds leading to an in-person finals event included as an integral part of the annual CyberMaryland Conference held each October in Baltimore. During the qualification rounds (typically spanning a three-day period), teams of up to six players download a specified virtual machine "target" that has been preconfigured with numerous vulnerabilities that must be identifed and fixed within a six-hour scoring window. A similar process is used for the second qualification round, although the target and objectives will change depending on the division—for example, high school teams may face a different operating system, and college/professional teams may encounter a challenge requiring forensics knowledge and the ability to successfully report their findings to the referees for scoring evaluation. Following the qualification rounds, the top eight teams in each division are invited to compete in the finals.

For the finals, high school teams must defend several servers from active attack by an onsite Red Team while simultaneously repairing any vulnerabilities discovered; college and professional teams defend a more complex set of servers while at the same time attempting to "capture"—and then defend—other servers they discover as part of a modified "Capture the Flag" game scenario. To help provide a realistic cybersecurity threat environment for players, the MDC3 gaming platform scores teams based not only on their ability to identify and fix vulnerabilities but also on how well they keep the vulnerability fixed over time. Thus, if a fixed vulnerability is re-exploited later in the day, the team will start losing points until they discover and remedy the situation. Consequently, the scoring process adds to the realistic flavor that the competition provides during gameplay—meaning that teams must embrace a proactive and ongoing cybersecurity posture instead of the commonly held "find-fix-and-forget" mentality found in the operational world. How teams successfully achieve this outcome depends on their ability to coordinate responsibilities, delegate tasks, prioritize actions, and apply other professional "soft skills" within skilled technical operations during gameplay.

Although it is still too early to determine the effectiveness of cybersecurity competitions in providing long-term meaningful value to the cybersecurity workforce, the sheer number of cyber challenges like MDC3 suggests they are considered useful tools in meeting that goal and promoting the cybersecurity discipline more generally.

Observations and Lessons Learned

Having briefly described the organization of the Maryland Cyber Challenge, I will now reflect on the past four years' competitions to offer readers key observations and insights that may assist in planning, marketing, and running their own cybersecurity competitions.

Fostering Gender Diversity

Perhaps the most striking observation about the Maryland Cyber Challenge is the lack of gender diversity among participants—something unfortunately representative of the cybersecurity profession as well. Meaningfully addressing this situation in both the cybersecurity and broader STEM fields remains an ongoing and prominent concern for schools and employers alike. Much continues to be written and discussed about the ongoing issue of gender equality in computer science [5, 6], but if the educational and professional communities embrace cyber competitions as a way of developing computer security practitioners now, they must also be used to facilitate a more diverse and gender-balanced workforce in the future.

One way to assist in reaching this goal is to ensure that male-dominated clubs and team environments are collegial, tolerant, and foster a culture that does not condone gender discrimination or harassment. Organizations that mentor girls and women interested in cybersecurity or STEM-related fields also play important roles in helping narrow the gender gap in computer science and cybersecurity education. Examples of such groups and programs include the UMBC Center for Women in Technology's (CWIT) "Bits and Bytes" program for high school girls interested in engineering and IT fields and the nonprofit Women's Society of Cyberjutsu (WSC), whose members (current cybersecurity practitioners) regularly teach girls and women about cybersecurity topics and practices via evening seminars, weekend workshops, and summer camps. Although much remains to be done in this area, ultimately each individual must be known, respected, mentored, and utilized appropriately and fairly based upon their talents and capabilities as a member (or potential member) of their desired profession or field.

Cheating

A significant issue facing cyber competition organizers is cheating. For MDC3, this is a concern both during the distributed qualification rounds (conducted unsupervised at a team's own location) and in the finals. To address these concerns, one college team advisor suggested having unaffiliated third-person monitors present during each team's distributed qualification rounds or implementing Web-based video surveillance to monitor the room where the teams were working. In 2014, that same advisor reported that one of his two teams competing in the finals communicated with his other team on technical items regarding the competition. Although initiated with no malicious intent, the conversation led the first team captain to revisit his own team's work based on information not previously considered. Was this inconsequential conversation akin to intentional cheating through social engineering? When an organization has multiple teams competing, should its teams be prohibited from discussing between them anything about the competition? Given logistical and other resource considerations during distributed qualification rounds found in MDC3 and other competitions, it is likely that such prohibitions are unenforceable, and proposed solutions to monitor teams remotely may not be practicable without significant volunteer, financial, and technical resources. Regarding this particular incident, the team's advisor conducted an internal inquiry and kept MDC3 organizers informed of the situation—ultimately, the team in question was allowed to compete as planned since there was no rule prohibiting teams from the same club at the same university from talking to each other.

During an in-person finals competition event, cheating is even more difficult to ascertain and/or counter: coaches, spectators, teammates, and supporters may develop ways of signaling information to competitors from the sidelines; participants may "bump into" advisors or supporters while going to and from the restroom (if located outside of the competition area); or, in perhaps the most egregious example of cheating witnessed at MDC3, an acquaintance may be positioned outside the competition area with a laptop and mobile phone ready to look up solutions to problems and relay them to the team inside the competition area. Overcoming cheating at in-person events requires not only a degree of trust in the teams and their advisors to abide by the rules, but also active patrolling and monitoring of the area in the immediate vicinity of the competition floor by staff to discover any possible indicators of cheating.

In terms of cheating, although cybersecurity competitions attempt to provide realistic environments for players, they are still only games—and games require a functional gaming environment for the competition to take place within. Therefore, "cheating" at cyber competitions also can include actions taken by participants to attack or disrupt the competition infrastructure (e.g., unplugging routers, DDoS attacks on network connections, and disabling scoring agents or required services on servers) during gameplay to prevent other teams from playing. Minimizing these types of gameplay risks include declaring the game infrastructure itself off limits as part of the competition rules of engagement and deploying network logging capabilities to help facilitate investigation into alleged attempts to "break" the game during play.

Unfortunately, given the nature of the competition, available technology, and potential limitations of facility layouts, it may not be possible to eliminate all sources of cheating during the event. In response to these concerns, although MDC3 never disqualified a team for cheating, it reserved the right to do so under a "one warning and you're out" policy. In such situations, the competition referees (the White Team) would consult with the teams in question, game engineers, and review network log data to determine whether a violation took place. During the first four years of MDC3, there were three warnings issued to teams during the MDC3 finals, but none resulted in disqualification or ejection from the competition.

Determining "Student" Standing

Although many cyber competitions are intended primarily for high school and college students, uneasy situations may arise in establishing what constitutes a "student" vis-à-vis competition objectives. For example, a person may be a highly trained cybersecurity professional at work but also enrolled in a part-time academic program in the evening as a (non-traditional) "student"—however, even though a person is indeed a "student," should he compete in the same division as other "students" with limited or no professional industry experience? In these contexts, other competitors may believe, rightly or wrongly, that some teams are populated with "ringers" who provide an unfair advantage. By contrast, could a motivated high school student compete on a college team in that division, even though she is technically a high school student? To preclude such perceptions or confusion, competition organizers should be mindful of what constitutes a "student" in their event, be flexible in how they approach establishing participant identity and eligibility for the competition, and ensure that these criteria are well-known in advance to all involved. Failing to do that may invite unnecessary drama during the event.

Proactive Communications and Outreach

Perhaps the most important things facilitating a successful cyber competition are the communication and customer service skills of the organizers. Not only is it crucial to set and manage participant expectations appropriately before, during, and after the competition, but when problems in execution inevitably occur, it is essential that teams are informed regularly in an objective and confident manner. Proactive and regular updates to teams (e.g., via email or Twitter) can reassure them that their concerns are noted and that the event organizers are actively working toward a resolution.

For example, in MDC3's inaugural year, a minor earthquake in San Diego created a sinkhole that disrupted communications links to the datacenter containing the MDC3 game environment less than an hour before the start of the first scored qualification round where 35 teams were standing by to compete. By providing regular updates to competitors (including projected estimates regarding repairs and/or when to expect the next situation update), the competition schedule was modified, and despite slipping the exercise start time nearly 48 hours, participants were able to plan accordingly and the competition went forward.

Cyber challenge organizers must never be accused of failing to keep teams informed or being unresponsive to their requests and inquiries—no matter how mundane or inconsequential. Good proactive communication is essential for all types of cyber competitions but is particularly important when dealing with high school students given the typical impulsive nature of adolescent students.

Well ahead of the competition, most organizers publish and/or otherwise inform teams about the rules governing gameplay—and also remind teams of the rules prior to the start of play. However, if any changes are made to the posted rules, they must be promulgated promptly and publicly to all teams. Failing to announce changes to the rules or gaming environment quickly (e.g., "Target #3 is disqualified for all due to unspecified technical problems; no points will be awarded for any work done on Target #3.") may lead to confusion, lost time, or anger exhibited by teams not aware of the change.

Avoiding Vulnerability "Conditioning"

In terms of training and educating students on cybersecurity practices, one of the key characteristics of MDC3 also is one of the most frustrating to participants. Specifically, MDC3 does not disclose what vulnerabilities are present or used for scoring on competitor systems, even after the scores are calculated. For example, if a team only found half of its assigned vulnerabilities during the qualification rounds, frequently they will inquire which vulnerabilities they missed so that they "can learn how to find and fix them" in the future. However, computer security vulnerabilities can manifest in many different ways and yield similar effects; therefore MDC3 organizers do not want to condition teams into believing that a certain vulnerability could only appear as it did during the competition. This policy is revisited regularly by event organizers but as of 2015 remains in place.

External Internet Access during Finals

During the in-person finals, another area of possible controversy regards access to the public Internet during gameplay. For MDC3, although teams were free (and expected) to use the Internet to research vulnerabilities and solutions during the distributed qualification rounds, during the onsite finals teams either had no or extremely limited Internet access (e.g., a shared and paltry 256K bandwidth assigned for the entire competition network) as a way of discouraging participants from using it during gameplay. This was done to reduce distractions such as social media use and to prevent cheating by teams planning to pre-position scripts or other tools on private external servers to gain an unfair advantage.

To compensate, the MDC3 game environment includes an internal patch server that allows teams to download whatever Windows or UNIX updates they believe are necessary to harden their systems and ensure availability. In cases where a team wants a particular tool or patch that is not available (such as a free, open-source tool like nmap), it may initiate a request through the White Team, who in turn discusses the request with the competition referees; if the request is granted, the game engineers will acquire the files in question and place them on the internal update server while the White Team announces to all participants that the new files are available. This ensures that no team has an unfair advantage in terms of software or technical resources. Of course, to help prevent cheating (which may include external access to the Internet), MDC3 maintains a "no mobile device" policy on the competition floor during the finals; however, it allows teams to use paper-based resources such as reference books, notebooks, or printouts they wish to bring to the event.

Cyber Challenges as Bragging Rights

Regardless of student or professional status or amount of prize money won, involvement with and/or winning a cyber competition is considered an attractive activity to list on a resume to demonstrate operational commitment to cybersecurity. Indeed, participation in cyber competitions is an attractive factor when corporate recruiters evaluate students for internships or other entry-level positions. Similarly, professionals competing in such events "on company time" have a strong interest in proving their skills to colleagues and supervisors, often with the strong support of senior leadership: for example, in 2011, the first-place MDC3 professional team was granted entry to the global Cyberlympics finals taking place the following week—their CEO offered strong support and authorized additional travel and time away from the office while they were onstage receiving their MDC3 prize.

As such, competition organizers should be prepared to generate award certificates, participation letters, or other "proof" of a participant's involvement beyond the awarding of any trophies, plaques, or bestowed "bragging rights"—which may include working with local media on stories profiling individual participants, teams, their schools/employers, or their preparations for the competition [7]. To support these efforts, competition organizers should maintain excellent records of scores, scoring criteria, and their interactions with teams that can serve as references if/when questions arise over competition outcomes.

"Unknown Unknowns" and the Competitive Spirit

As with any large event, competitive or otherwise, there will be spontaneous issues, problems, concerns, and situations that organizers did not consider during the planning process. This is particularly problematic when planning cyber competitions for elementary and high school participants, where organizers not only must coordinate competition items across multiple schools, school districts, and states, but may not be aware of every conceivable special situation or resource limitation (i.e., policy, financial, infrastructure, scheduling, or instructional) facing those schools. As a result, unanticipated incidents may present the event in a less-than-favorable way to school administrators or parents.

For example, during the inaugural MDC3 in 2011, one high school team discovered they could not maintain steady access to the game environment during the first qualification round due to an updated configuration of their school's firewall following the earlier practice rounds. Upon encountering this problem, and unbeknownst to their faculty advisor, the team simply modified their school's firewall and continued competing in the round. While their ingenuity allowed the team to move into the second qualification round, the team's faculty advisor (and computer science teacher) was forced to defend the team's actions to the Principal and school IT Manager the following week—at which point school administrators became aware of the nature of (and skills needed for) such competitions. Although the incident was resolved without punishment, it exemplifies some of the "unknown unknowns" that can arise when highly motivated young participants embrace the competitive spirit. Thus, when preparing for competitions, regardless of student or professional status, teams should ensure that all competitors are cognizant of any local computer use or security policies and coordinate their actions with the appropriate IT staff well in advance of the event. Here again, proactive and ongoing communications with all involved can minimize the potential for confusion or unfavorable views on either the team or competition.

These are some of the more noteworthy observations and recommendations emerging from the first four years of the Maryland Cyber Challenge. Although no organizing team can predict every contingency, appropriate prior planning, proactive communications, diligence in maintaining a fair and diverse competition environment, a degree of objective operational flexibility, and effective management of the expectations of all involved can help facilitate successful and meaningful events.

Final Thoughts and Admonitions

As a partial retrospective, I have shared some key observations and lessons learned from the first years of the Maryland Cyber Challenge (MDC3) in an attempt to offer useful advice to current and future competition organizers in helping them develop and conduct successful events of their own.

Cyber competitions are a useful tool for the cybersecurity industry. However, in developing the cybersecurity workforce, we must be mindful that cybersecurity competitions tend only to emphasize the demonstration and application of specific hands-on skills to address technical symptoms of current problems versus developing the fundamental or interdisciplinary knowledge to remedy, if not prevent, their root causes [8]. While certainly necessary for success in cyber competitions, and quite useful in the world of technical cybersecurity operations, the knowledge and attributes needed by well-rounded security practitioners—indeed, professionals in any field—must extend beyond the (albeit important) technician-level skills that cyber competitions like MDC3 inculcate. As suggested in a recent Pew research survey [9] and by political observers [10], personal characteristics such as self-reliance, inquisitiveness, critical thinking and analysis, teamwork, strong communication skills, adaptability, excellent organizational or management capabilities, understanding of the theoretical foundations of technology, and situational awareness maintained across an interdisciplinary spectrum are just as, if not more, important as technical skills to a person over the length of their professional career.

Indeed, developing and/or possessing excellent technical skills may qualify a person for a series of jobs that earn a paycheck, fill critical roles in industry, and help meet the politically expedient goal of workforce development—however, a career in cybersecurity involves a far broader set of knowledge, skills, and abilities. Therefore, while cyber competitions are popular events that encourage many toward or further into a career in cybersecurity, we must remember that cybersecurity itself is an interdisciplinary field—and that not all positions in the cybersecurity realm will require expert technical skills honed through competition alone.

References

[1] Education Advisory Board, "Multi-Track Cybersecurity Pathways" (Industry Futures Series), Washington, DC, 2014: mirrored at http://www.csee.umbc.edu/~rforno/EAB-Cyber-2014.pdf.

[2] Sarah Halzack, "Report Finds D.C. Area a Hotbed for Cybersecurity Jobs," *Washington Post,* March 8, 2014: retrieved from http://www.washingtonpost.com/business/capitalbusiness/report-finds-dc-area-a-hotbed-for-cybersecurity-jobs/2014/03/08/1b72ff1e-a560-11e3-8466-d34c451760b9_story.html.

[3] Fort Meade Alliance, "Epicenter of Cyber Security": retrieved from http://www.ftmeadealliance.org/mbc/doing-business/epicenter-of-cyber-security.

[4] CyberFed, "About Us," 2015: retrieved from http://cyberfed.org/about.html.

[5] D. Beede, T. Julian, D. Langdon et al., "Women in STEM: A Gender Gap to Innovation," US Department of Commerce Economics and Statistics Administration, 2011: retrieved from http://www.esa.doc.gov/sites/default/files/womeninstemagaptoinnovation8311.pdf.

[6] American Association of University Women (AAUW), "Solving the Equation: The Variables for Women's Success in Engineering and Computing," 2015: retrieved from http://www.aauw.org/research/solving-the-equation/.

[7] Lauren Loricchio, "Catonsville High Cyber Team Members Prepare for the Future," *Baltimore Sun*, October 28, 2014: retrieved from http://www.baltimoresun.com/news/maryland/baltimore-county/catonsville/ph-ca-cyber-security-1022-20141028-story.html.

[8] D. Burley, "An Interview with Gene Spafford on Balancing Breadth and Depth in Cybersecurity Education," *ACM Inroads*, vol. 5, no. 1, pp. 42-46.

[9] Sara Kehaulani Goo, Pew Research Center Blog, "The Skills Americans Say Kids Need to Succeed in Life," February 19, 2015: retrieved from http://www.pewresearch.org/fact-tank/2015/02/19/skills-for-success/.

[10] Fareed Zakaria, "Why America's Obsession with STEM Education Is Dangerous," *Washington Post,* March 26, 2015: retrieved from http://www.washingtonpost.com/opinions/why-stem-wont-make-us-successful/2015/03/26/5f4604f2-d2a5-11e4-ab77-9646eea6a4c7_story.html.

SYSADMIN

/var/log/manager
I'm the Manager, This Is My Job

ANDY SEELY

Andy Seely is the Chief Engineer and Division Manager for an IT enterprise services contract, and is an Adjunct Instructor in the Information and Technology Department at the University of Tampa. His wife Heather is his PXE Boot, and his sons Marek and Ivo are always challenging his thin-provisioning strategy. andy@yankeetown.com

I've written "/var/log/manager" for the past two years. The concept was to open a window into how technical managers think and why they act the way they do, and to offer help and guidance to young sysadmins trying to grow in their careers. This is my last "/var/log/manager," and I hope you've enjoyed reading as much as I've enjoyed writing. I'd love to hear from you. What are your thoughts, ideas, and problems with respect to these issues?

Over the last 11 articles, I have covered themes of communication, understanding, leadership, career management, prioritization, and problem-solving. This last "/var/log/manager" entry is my manager's playbook.

Communicate so people can actually understand [1]. Communication is surprisingly difficult, given that it's the thing people give the appearance of doing more than anything else. Communicating across boundaries of expertise and understanding is an extra challenge. We assume that a common spoken language like English sets a standard for common understanding, but domain knowledge of a highly technical sysadmin topic is usually not evenly distributed. Communication is improved when you understand the other person and try to hear the sound of your own voice through their ears. Accept that communication isn't about the shipping, it's about the receiving and getting the message right so it can be received.

Blame is a counterproductive luxury [2]. What is the purpose of your systems? To provide whatever resource they're built for. What is your purpose as a sysadmin? To maximize the effectiveness of the systems. In highly complex environments, things go wrong. Add nondeterministic meat-based systems like people and more things are available to go wrong. I consider the whole thing as a holistic system, with silicon, virtual, and meat nodes. When a computer fails due to lack of resources or poor configuration in the network, we don't question fixing it. When a sysadmin fails due to lack of resources, time, knowledge, or even judgment, it's easy to blame first. Understand why failures happen and correct the root causes. Sometimes a computer needs to be replaced, and sometimes a person needs to be replaced, but focusing on blame first means you're not focusing on understanding the complexity of your system.

Don't believe your own hype [3]. We're sysadmins, a term I use as a synonym for "rock-star-awesome people." It can be easy to get caught in a hype cycle and start performing for an audience rather than focusing on the things that matter to the business. Rock stars get to be rock stars through hard work, putting in the time, and sweat equity, and they don't stop doing those things when they get to be stars. Prioritize the little things, the mundane necessities, and the core functions that make your business successful, and let someone else be the rock star.

Performance tuning and fault isolation analysis works for organizations as well as for systems [4]. When a system isn't performing well, you analyze and troubleshoot. What are the key performance indicators? Where are the bottlenecks? What are the producers and consumers? The threat vectors and attack surface? Baseline, then measure, adjust, monitor, repeat until there's a new baseline that meets requirements. I think it's funny to talk about systems as made up of both silicon and meat, but the people-based systems we call organizations may be approached, understood, and tuned in much the same way. Understand your systems and optimize them, regardless of what they're made of.

Help people understand and overcome problems [5]. I like to remind people that their jobs are hard, and that's why we have them to do the jobs. If the jobs were easy, we'd have someone else. Find ways to help people through their roadblocks so they can get things done. Sometimes that's a hard push, sometimes it's removing a blockage, and sometimes it's just getting in their heads with an idea that they can use in their own way to navigate a problematic terrain.

Do the little things so they don't become big things [6]. There's nothing glamorous about doing a time card or taking a certification test. There's also nothing glamorous about getting fired over something so trivial. Do the easy, mandatory stuff first. Don't make it your manager's job to make you do it.

Get in each other's heads and see the world from someone else's eyes [7]. People will work best when they bring knowledge, skills, and abilities and they're met in the middle with guidance, trust, and support. It's very easy to allow a gulf to come between a sysadmin and a manager and to assume that they don't understand each other just because they have different priorities. The gulf grows by itself, without any help. It's important to keep the gap in mutual understanding as small as possible, and shrink it whenever you can.

Finding ways to categorize people is just a step towards understanding them [8]. The goal of any business is supported by computing systems, which are in turn supported by sysadmins. Two equally great sysadmins may have very different abilities. Making an over-generalization about people can be misguided if it's the only thing you do, but a first step towards understanding someone can be to assess how they learn and how they work. This is a round pegs/round holes situation on the surface, but at its core the real goal is to put an individual sysadmin in a role that maximizes their capabilities.

Don't treat outliers like they're the new normal [9]. In large organizations and large systems, the chances are high that something unexpected will happen at some point. If something really weird, out of the ordinary, and stupid happens, remind everyone around you that the most important thing is "out of the ordinary." It's incredibly wasteful to re-engineer a complex process because there's one outlier, but it is human nature to focus on the outlier. As we say in the South, "The high nail gets the hammer." Conversely, if you find yourself explaining away a lot of weird events as outliers, you might want to reassess your baseline of normal.

Understand why people come to work, and give them reasons to keep coming to work [10]. Each employee has a unique situation and a personal set of motivators that they themselves may not truly understand. Taking the time to codify why people come to work is the first step. Figuring out which parts you can actually influence is the second step. Actively doing things to positively influence those things is the approach, and the goal is to keep smart people happy and productive.

Don't let your information flow become your enemy [11]. Understand your own information inputs, outputs, and repositories. Understand and articulate your own personal priorities. Make information flow be your tool rather than your nemesis. Always prioritize people first, and give your attention to the person you're talking to.

Be the manager, but don't forget to be the leader. The difference between being the manager and being the leader is simply where you stand: managers are in the back directing, while leaders are in the front leading. Managers tell you what to do, and leaders inspire you to do it. The best are those who are both at the same time.

I will leave you with a last thought: when you've been following leaders, mentors, and guides as you moved through your career, one day you'll realize that you don't have anyone leading you anymore. Just look over your shoulder. You'll find a line of people who have been following you all along and you didn't even realize it. At that moment, you become the leader you were looking for.

References

[1] "Message Not Received," *;login: logout*, January 2014: https://www.usenix.org/publications/login-logout/january-2014-login-logout/seely.

[2] "Let's Find Someone to Blame," *;login:*, vol. 39, no. 2, April 2014: https://www.usenix.org/publications/login/april14/seely.

[3] "Rock Stars and Shift Schedules," *;login:*, vol. 39, no. 3, June 2014: https://www.usenix.org/publications/login/june14/seely.

[4] "When Technology Isn't the Cause of a Technical Problem," *;login:*, vol. 39, no. 4, August 2014: https://www.usenix.org/publications/login/august14/seely.

[5] "Parables of System Administration Management," *;login:*, vol. 39, no. 5, October 2014: https://www.usenix.org/publications/login/october-2014-vol-39-no-5/varlogmanager-parables-system-administration-management.

[6] "Career Preventative Maintenance Inspections," *;login:*, vol. 39, no. 6, December 2014: https://www.usenix.org/publications/login/dec14/seely.

[7] "Daily Perspectives for the Sysadmin and the Manager," *;login:*, vol. 40, no. 1, February 2015: https://www.usenix.org/publications/login/feb15/seely.

[8] "A Generational Theory of Sysadmins," *;login:*, vol. 40, no. 2, April 2015: https://www.usenix.org/publications/login/apr15/seely.

[9] "Surreal Management Situations," *;login:*, vol. 40, no. 3, June 2015: https://www.usenix.org/publications/login/june15/seely.

[10] "Incentivizing Smart People," *;login:*, vol. 40, no. 4, August 2015: https://www.usenix.org/publications/login/aug15/seely.

[11] "How Technical Managers Tell Time," *;login:*, vol. 40, no. 5, October 2015: https://www.usenix.org/publications/login/oct15/seely

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Publish and Present Your Work at USENIX Conferences

The program committees of the following conferences are seeking submissions. CiteSeer ranks the USENIX Conference Proceedings among the top ten highest-impact publication venues for computer science.

Get more details about these Calls at **www.usenix.org/cfp.**

CoolDC '16: USENIX Workshop on Cool Topics on Sustainable Data Centers

March 19, 2016, Santa Clara, CA

Paper submissions due December 15, 2015

The 2016 USENIX Workshop on Cool Topics in Sustainable Data Centers (CoolDC '16) is a forum to disseminate results and stimulate further cutting-edge research in quantitative design, evaluation, and research methods for sustainable data centers. The goal of the workshop is to become a venue where experts in sustainable energy systems, data center physical infrastructure, networking and server architecture, cloud computing, and internet-scale applications can come together to exchange ideas on how to maintain and improve the sustainability of warehouse-scale computer infrastructure.

USENIX ATC '16: 2016 USENIX Annual Technical Conference

June 22-24, 2016, Denver, CO

Paper and Talk submissions due February 1, 2016

USENIX ATC '15 will again bring together leading systems researchers for cutting-edge systems research and unlimited opportunities to gain insight into a variety of must-know topics, including virtualization, system administration, cloud computing, security, and networking.

Authors are invited to submit original and innovative papers to the **Refereed Papers Track**. We seek high-quality submissions that further the knowledge and understanding of modern computing systems with an emphasis on implementations and experimental results. We encourage papers that break new ground, present insightful results based on practical experience with computer systems, or are important, independent reproductions/refutations of the experimental results of prior work.

Industrial practitioners are invited to submit talk proposals to the **Practitioner Talks Track**. This track seeks presentations about practical solutions and challenges to significant real-world issues facing industrial practitioners. It will provide a unique venue for industrial and academia participants to exchange ideas and experiences.

HotCloud '16: 8th USENIX Workshop on Hot Topics in Cloud Computing

June 20-21, 2016, Denver, CO

Paper submissions due March 8, 2016

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/emerging "hot" trends in this important area. While cloud computing has gained traction over the past few years, many challenges remain in the design, implementation, and deployment of cloud computing.

HotCloud is open to examining all models of cloud computing, including the scalable management of in-house servers, remotely hosted Infrastructure-as-a-Service (IaaS), infrastructure augmented with tools and services that provide Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

HotStorage '16: 8th USENIX Workshop on Hot Topics in Storage and File Systems

June 20-21, 2016, Denver, CO

Paper submissions due March 10, 2016

The purpose of the HotStorage workshop is to provide a forum for the cutting edge in storage research, where researchers can exchange ideas and engage in discussions with their colleagues. The workshop seeks submissions that explore longer-term challenges and opportunities for the storage research community. Submissions should propose new research directions, advocate non-traditional approaches, or report on noteworthy actual experience in an emerging area. We particularly value submissions that effectively advocate fresh, unorthodox, unexpected, controversial, or counterintuitive ideas for advancing the state of the art.

Submissions will be judged on their originality, technical merit, topical relevance, and likelihood of leading to insightful discussions that will influence future storage systems research. In keeping with the goals of the HotStorage workshop, the review process will heavily favor submissions that are forward-looking and open-ended, as opposed to those that summarize mature work or are intended as a stepping stone to a top-tier conference publication in the short term.

OSDI '16: 12th USENIX Symposium on Operating Systems Design and Implementation

November 2-4, 2016, Savannah, GA

Abstract registration due May 3, 2016

The 12th USENIX Symposium on Operating Systems Design and Implementation seeks to present innovative, exciting research in computer systems. OSDI brings together professionals from academic and industrial backgrounds in 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.

www.usenix.org/cfp

Interview with Darrell Long

RIK FARROW

Dr. Darrell D. E. Long is Professor of Computer Science at the University of California, Santa Cruz. He holds the Kumar Malavalli Endowed Chair of Storage Systems Research and is Director of the Storage Systems Research Center. His broad research interests include many areas of mathematics and science, and in the area of computer science include data storage systems, operating systems, distributed computing, reliability and fault tolerance, and computer security. He is currently Editor-in-Chief of *ACM Transactions on Storage*. darrell@soe.ucsc.edu

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

The Intel/Micron announcement of XPoint 3D in July 2015 really got my attention [1]: finally, a vendor will start shipping a form of non-volatile memory (NVM) that's not NAND flash. XPoint 3D promises to be byte addressable, faster, more durable, and require lower power than any form of flash today. The downsides (there are always downsides) will be that XPoint 3D will be more expensive and have less storage capacity when it appears in early 2016.

Having byte-addressable NVM will have impacts on the way computers are designed and operating systems and software are written. If this technology proves to be everything that Intel and Micron are promising, it might change everything about the systems we are familiar with. At the very least, XPoint 3D would become a new tier in the storage hierarchy.

I asked around, trying to find someone I knew in our community who could address this topic from a file system and storage perspective. The timing was terrible, as all of the people I asked (who responded) were busy preparing FAST '16 papers for submission, and with two deadlines at about the same time, you can guess which one is the more important.

Darrell Long, a professor at UCSC, took up my challenge, even though he too was busy on an overseas trip, as well as supervising papers to be submitted to FAST '16. Long has experience in both storage systems and operating systems, and seemed like the right person to talk to about this development.

Rik: I recently heard about a new type of NVM developed by Intel and Micron and in production. I've been hearing talks about how a technology like this could result in large changes in the designs of systems and operating systems for many years.

Darrell: I don't know a lot about the details of the technology, and was waiting to get home from travel to sit down with Intel for a technical briefing on the details, but I agree with you 100% that this may in fact be a game-changer. Ethan and I wrote a paper on this about 14 years ago, when there was hope for MRAM (before physics got in the way) [2].

One of the key points is that unlike flash, this *will* be on the memory bus. We will have persisted memory that is lower power, denser, and unfortunately slower than DRAM. But it will be byte addressable. That means all the tricks we have developed over the years for packing data structures into blocks go away—at least at that level.

My belief is that files do not go away, they are simply too useful. I think it would be a mistake to tie an object, say a photograph, to an application (as Apple loves to do with the iPad). Objects may become first-class entities, and you can then think of applications as operators that perform transformations on them. There will be a lot of them, so we still need naming and protection, and we still need long-term storage—so I do not see spinning rust disappearing, at least not for quite a while, but it may move back into "archive" where we keep our named objects. And as before with disk, we will still need to worry about mapping memory addresses to some (in this case) higher level representation; unlike Word in the old days, you can't just dump persistent memory to disk and load the image.

Rik: MRAM never happened, and XPoint 3D will be slower than DRAM, initially by a factor of 100,000, although the claim is that this will come down to 1000 times slower than DRAM.

Darrell: Physics got in the way of MRAM, and phase change memories were not yet a thing when we wrote the paper [2]. So MRAM will never happen, except for low-density stuff that needs to be radiation hard (but phase change is also radiation hard).

I think the key issues for this new technology are byte-addressability and its speed relative to the others. For high performance computing, power consumption will also be huge. Most people do not realize it, but the vast majority of energy comes from moving the bits, not the computation. If you look at a processor die, it's all cache and bit movement. The ALU and FPU are tiny parts of it these days.

Having byte-addressable persistent memory on the memory bus is, I think, a game changer. I am not sure how it will all shake out. But the usual ideas of never having to fully reboot (but we need to retain that ability due to the crappy software that gets written) will certainly come up.

I think the key things that we need to think about are how we manage our data: how do we name it, find it, and protect it? The file model works pretty well, and we may not want to just throw it away. But we can lose a lot of its strictures.

Rik: I agree, moving bits is expensive, and have been looking at dies (well, masks for dies) since the early '80s. Now, it's mostly cache and the parts that determine whether the right line is present or not.

As for crappy software, even geniuses can't write perfect software, and most people who write software aren't geniuses. Someday, perhaps software systems will write software, but even then there might be memory leaks, accidental corruption of data structures, and so on.

And will all of the strictures on file system design go away? I think file systems will still use locks, write metadata before data is written, and so on. NVM file systems will need to be different, or should be different, for handling media that is byte rather than block addressable. That might be the interesting bit, given examples like NTFS, where MS systems programmers decided to have irregularly sized data structures, as they did in their in-memory logging system, and had terrible trouble with reliability and in the performance and reliability of their logging system.

Darrell: I haven't completely thought it through yet, but I think that the role of file systems will change. And I think that consequently they will get simpler. Consider the object abstraction we have been pushing for about 20 years (Swift [3] was 1991), and it is finally getting traction through Ceph and Seagate's Kinetics. A lot of the low-level stuff you can just push off onto the device, and let the higher-level file system worry about naming, load balancing and distribution, and protection (though some of that must be at the object level too).

Now look at the persistent memory. If the density is high enough, we really don't need the low-level parts of the file system on, say, your laptop. We already have flash, although that pretty much pretends to be a fast disk. But we will still need the naming and protection; when we want to back up our system, we will still want to use something like files. We will be taking byte-addressable memory and mapping it to block storage, be it flash, disk, tape, or the long-promised holostores.

So the locking and so forth will still be there, but it will change. It will be more like shared memory, and in the beginning before programming models really change I would expect a lot of mmap() kinds of things to happen. The back-end file systems may get a lot simpler, since they will probably not need to support range locking or update in most cases. Remember Tanenbaum's "Bullet" file system? [4] They could very well end up like that, with immutable files that get written in one fell swoop.

References

[1] Intel Micron XPoint memory announcement: http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology; analysis by Anandtech: http://www.anandtech.com/show/9470/intel-and-micron-announce-3d-xpoint-nonvolatile-memory-technology-1000x-higher-performance-endurance-than-nand.

[2] Ethan L. Miller, Scott A. Brandt, and Darrell D. E. Long, "HeRMES: High-Performance Reliable MRAM-Enabled Storage," *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII),* Elmau, Germany: IEEE, May 2001, pp. 83–87: ftp://ftp.soe.ucsc.edu/pub/darrell/HotOS-Miller-01.pdf.

[3] Luis-Felipe Cabrera and Darrell D. E. Long, "Swift: Using Distributed Disk Striping to Provide High I/O Data Rates," *Computing Systems,* vol. 4, no. 4 (1991), pp. 405–436: https://www.usenix.org/publications/compsystems/1991/fall_cabrera.pdf.

[4] Robbert van Renesse, Andrew S. Tanenbaum, Annita Wilschut, "The Design of a High-Performance File Server," *9th International Conference on Distributed Computing Systems,* 1989, pp. 22–27: https://static.aminer.org/pdf/PDF/000/297/552/the_design_of_a_high_performance_file_server.pdf.

HISTORY

Offshoots

STUG and LISA

PETER H. SALUS

Peter H. Salus is the author of *A Quarter Century of UNIX* (1994), *Casting the Net* (1995), and *The Daemon, the Gnu and the Penguin* (2008). peter@pedant.com

This column is brought to you by the letter K. K as in Kernighan and K as in Kolstad. Of course, others were involved: Bill Plauger, Deborah Scherrer, Dennis Hall, Joe Sventek. Were you alive in 1976? In 1987? Well, pull up your chairs and learn ancient lore.

Software Tools by Brian Kernighan and P. J. ("Bill") Plauger bears a copyright date of 1976. I own two copies: one was given to me as a Christmas gift by Lou Katz in 1978. It's full of notes. The other was purchased in the mid-1990s when I was working on *The Handbook of Programming Languages* (4 vols., 1998). I still consider it the very best book on programming I've ever read. Andy Tanenbaum (world-renowned computer scientist of the Vrije Universiteit in Amsterdam) thought so, too. He left a copy at Lawrence Berkeley Labs (LBL), telling Debbie, "You might be interested in this." She was.

"I thought it was wonderful," she told me. "At the same time that Andy mentioned the tools book, Dennis Hall discovered it. And Dennis got me involved and got the PI to approve using my time. So over the weekend I started to implement all the tools. It was great!"

It was that fast, too. Debbie and Dennis and Joe Sventek did it all.

"Remember," Mike O'Dell told me, "Debbie knew Brian, and he knew what they were doing, so he pointed people who asked about the tools to LBL. And that started the Software Tools User Group."

User group? Yes. A BoF was held at the USENIX meeting in 1979 in Toronto. Just over 100 attended. A newsletter was initiated. And in volume 1, number 2 of *Software Tools Communications* (November 1979) there was this announcement: "The next meeting of the Software Tools User's Group will be held January 29th in Boulder, Colorado."

Software Tools wasn't about UNIX, it was about philosophy and style. The late Dennis Ritchie told me, "The tool-using approach is powerful and intellectually economical, but it takes imagination to use."

Interestingly, Debbie, Dennis, and Joe realized just how powerful the tool concept was. They wrote a virtual operating system (VOS) that would serve as a pseudo-interface between the software tools written in Brian's Ratfor (Rational Fortran) and whatever OS was running. The paper, "A Virtual Operating System," appeared in *CACM* 23.9 (September 1980). In April 1996, Debbie, Dennis, and Joe were awarded the USENIX Flame for their efforts.

STUG waxed rapidly. In April 1980 it had over 2,000 members. And Debbie prepared a "Cookbook" (January 1981), which comprised "Instructions for implementing the Software Tools Package (as distributed by the Software Tools Users Group)." The tools? Oh, the tools ran (alphabetically) from **ar** (archive) to **xref** (make a cross-reference).

Software Tools Communications ceased publication after January 1986. But a decade later, USENIX began an annual award. "The STUG Award recognizes significant contributions to the community that reflect the spirit and character demonstrated by those who came together in the Software Tools User Group (STUG). Recipients of the annual STUG award conspicuously exhibit a contribution to the reusable code-base available to all and/or the pro-

usenix 40TH ANNIVERSARY 1975–2015

vision of a significant enabling technology to users in a widely available form." The STUG Award originated in a donation by Debbie Scherrer of the funds remaining after the group ceased activity.

LISA

Rob Kolstad, then at Convex in Texas and newly elected to the USENIX Board, mentioned the difficulties of large systems at a board meeting and, together with Alix (Max) Vasilatos, then at MIT, organized a "Large Installation System Administration Workshop," held in Philadelphia in April 1987. There were 55 attendees. I vividly recall Rob asking whether anyone "backed-up" 10 meg a night. About a third of those present stood. He then increased the number. There were still two at "over 100 meg."

In those days it was a lot.

The LISA15 event many readers attended was the 29th. I hope there will be a major celebration for the 30th in 2016.

When LISA began, no one even imagined "the cloud" or installations like those of Google or Amazon. The largest users were government and military. Today most of us have gigabytes on our desks. One can buy 5T for well under $200. What will "large" mean in another few years?

Thank you Brian and Bill; Debbie, Dennis, and Joe; Rob and Max.

Epilogue

At year-end, it's customary to predict what's coming. I'm reluctant to do that. Let's face it, we do a really lousy job at prediction—especially of the future. By the way, although that's frequently attributed to the late Yogi Berra and occasionally to Sam Goldwyn, it comes from a question and answer period during a seminar in Copenhagen where Niels Bohr laid out the fundamental nature of quantum physics. Included was the description of the Heisenberg uncertainty principle, which basically says that you can't predict where a particle will be at a specific place in time, or vice versa. The question that triggered the answer was: "What do you predict the influence of quantum physics will have on the world in the future?" and Bohr said, somewhat tongue in cheek due to the prominence of the principle, that "it is exceedingly difficult to make predictions, particularly about the future" (because we can't even know what the state of our situation is NOW, much less in the future).

Dr. Susan Calvin made her first appearance in "Robbie," but she wasn't in the version that appeared in *Super Science Stories* in September 1940. Asimov "adjusted" his history and inserted her in a revised version. But that wasn't all he revised: he moved the action from 1982 to 1998. As he died in 1992, I guess he didn't concern himself with shoving things further.

H. G. Wells originated moving sidewalks, but my guess is that most people first thought of them in Heinlein's "The Roads Must Roll," which appeared in *Astounding Science Fiction* magazine in 1940. Now, nearly every airport has a slow version of them (though when I was in Vancouver, half of them weren't running and in Portland, Oregon, none were). A roadway from Boston to New York to Philadelphia to Baltimore to Washington would be great. By the way, Heinlein's *Rocket Ship Galileo* appeared in 1947 and involves Nazis on the moon.

These are just to further illustrate that Bohr was right. Happy New Year!

Interview with Rob Kolstad

RIK FARROW

After earning his PhD, Rob Kolstad spent eight years at successful supercomputer startup Convex Computers as the Manager of Operating Systems. During his tenure there, he was elected to serve the first of his three terms on the USENIX Board of Directors. He joined Prisma Technica, a Colorado Springs-based startup whose mission was to construct a high-speed gallium arsenide SPARC processor, which led to two years with Sun Microsystems, followed by seven at the first Internet server company, Berkeley Software Design (BSDi). He was also a program manager at SANS, and later Executive Director of SAGE for USENIX. Most recently he has been an intellectual property consultant, working on projects like analysis of voting machine code and patent infringement investigations (including disk drives, flash components, embedded software reverse engineering, and DVD recording formats). Rob also volunteered for 20 years as the head coach of the USA Computing Olympiad, training pre-college students to represent the US at annual international programming contests held around the world. rob.kolstad@gmail.com

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

I believe I first met Rob Kolstad in the early '90s, during the time of the AT&T lawsuit against BSDi and the University of California. I was very interested in what was happening in that lawsuit, partially because I consulted for *UNIX World* magazine in those days, and largely because I believed in open source. Rob was the head of BSDi by then, and he arranged for me to receive versions of BSD as they were updated.

Rob also had history with USENIX by that time, having (among other activities) started LISA as a small workshop (see Peter Salus' article in this issue). Rob was the editor of *;login:* for many years too, as well as a popular speaker and conference organizer.

But there's surely more to his story, I thought, so I decided to see what I could elicit.

Rik: Tell us about your early days in computer science.

Rob: I was told "you must go to college" from the day Sputnik launched. My parents independently emphasized this starting from first grade when I mentioned standardized tests we had just taken. It was a given and as natural as being born or getting up in the morning. "You must go to college."

After spending half a dozen years as a very young programmer hanging out at the labs of the University of Oklahoma, I headed to SMU to work on one of the first computer science degrees. I followed some professors to Notre Dame for a master's degree in electrical engineering (all theory). From there it was six years at the University of Illinois, ending up with a thesis in the realm of distributed operating systems.

Convex Computer Corp. (then called Parsec Scientific Computers) was hiring OS folks to port UNIX to their new vector processor. As it turns out, I was one of the very few on the market at the time and thus joined them and learned how to play the startup game (this being the 1982 timeframe when supercomputers were the bubble instead of PCs or the Internet). Convex worked out well, going public and then increasing their share price.

I followed my boss to Colorado Springs to join Prisma Technica, who were going to manufacture a SPARC-compatible processor using gallium arsenide chips, which would have the then-blazing clock rate of 250 MHz. GaAs does have its challenges; the software group eventually ended up joining Sun Microsystems in order to ply their large system expertise for Sun's higher-end efforts.

After two years with Sun, I moved on to BSDi, whose goal was to commercialize Berkeley-style UNIX; it was my dream job. The lawsuit with AT&T over their misunderstanding of the provenance of BSDi's code cost over a million dollars and enabled the completely unencumbered Linux to gain its first foothold; the rest is history.

I then moved to work with conference-running organizations like USENIX and SANS before commencing independent consulting in the intellectual property world. Some of my projects there have included analyzing voting machine code, dissecting solid state disks in search of patent violations, and constructing the software to interpret and summarize high-speed bus traffic.

Rik: You started LISA in 1987 as the Large Installation System Administration conference. Can you provide some background about why you thought USENIX needed a conference about managing large numbers of UNIX systems? To be honest, I didn't even know there were large numbers of UNIX systems to manage in 1987.

Rob: I was privileged to attend USENIX conferences starting late in my graduate school career. It devolved in my circles that one's ticket to the conference was contingent upon publishing a paper there, and so that's what I did. I enjoyed the experience and worked on the things that seemed important: organizing late-night panels, helping with tasks, and generally slowly growing into a role of contributor (talks, tutorials, keynotes, chairing conferences, creating distribution tapes, etc.).

Early in my tenure at Convex Computer Corporation, USENIX founder Lou Katz suggested that I run for the USENIX Board of Directors. I wrote a platform statement that apparently resonated with the electorate and joined the Board.

I believe in "working boards" vs. those who use their contacts for fundraising (in the case of charities) or business promotion (in the case of corporations). I believe there was a general feeling that board members were in charge of presenting ideas for workshops, conference enhancements (including topics, tutorial ideas, and non-technical ideas), and generally creating the organizational plans which the main office then executed.

Of course, folks would often propose programmatic ideas in their own fields of interest, so I proposed a "Large Installation System Administrators' Workshop" (among others, including a supercomputer workshop that fared less successfully in the long-term). I was working as Manager of Operating Systems at Convex Computers; my job was to ensure that BSD UNIX ran seamlessly on the new Convex C1 affordable supercomputer. Even our organization of under 50 people was expanding into ever greater numbers of systems, and, of course, utilized large systems in order to ensure they worked when delivered to customers.

Some folks think back and wonder about how "large" a shop could syadmins in our marketplace administer. Here's an excerpt of the original announcement:

> [The] Large Installation System Administrator's Workshop [will] be held in Philadelphia, PA, on April 9th and 10th, 1987. This workshop will bring together system administrators trying to conquer UNIX's historical bias towards smaller systems. It is believed these administrators battle many of the same problems repeatedly and can share their unique solutions to some problems in order to avoid duplication of effort as UNIX grows to run in ever larger installations.
>
> System managers of shops with over 100 users (on one or several processors) will find this workshop particularly valuable....
>
> Some topics to be considered include: large file systems (dumps, networked file systems), password file administration (including YP), large mail system administration, USENET/News/Notes administration, mixed vendor (and version) environments, load control and batch systems, handy new utilities, and large LANs.

I seem to remember that, early on, one of the qualifications for "large" was "administration of over a gigabyte of disk storage." Times have changed!

The lack of unified tool sets, the complexity of the tools that did exist, and the emergence of new challenges all made large installations an interesting problem. Recall that this is back in the day when a system crash (which was not an infrequent occurrence) required running the fsck program to repair the disks before the system could be fully booted.

Rik: Can you tell us what's involved in coaching USACO participants? Do you help choose participants, create test problems, manage logistics, or do other work?

Rob: As the computer industry matured over the years, so did the USACO coaching techniques. In the earliest years, the problem was simply finding students who had the skills to participate. Later, there were more than enough students, so a camp was organized to perform training via lectures and labs (along with both technical activities like program AI-players for games and non-technical activities like ultimate frisbee), which led to a pair of five-hour on-site contests to select the representatives for the international contest.

A number of things became clear:

- Manual task grading was onerous, time-consuming, and had a potential for errors.
- A host of students could benefit from earlier exposure to the task types.
- Ultimately, the Web could provide a fertile training environment.

Thus, the coaches developed a grading system to run a contestants' program using a number of different data sets for input and provide results. While this might sound easy today, it was revolutionary in its time (hearken back to the days when people were mystified by UNIX, processes, the Web, and networking in general). That system ultimately ran several of the international competitions.

The coaches created test data for these tasks, and that became a challenge since erroneous test data leads to competitor complaints and a general tarnishing of reputation. Russ Cox (now Dr. Cox, at Google) led the way to "validators" whose job was to ensure that the test data conformed to the task description and was valid. Ultimately, the system could read the input descriptions and generate the validators automatically; this proved to be an extraordinary technique for ensuring that nonconforming data did not sneak into the grading process.

The automated grading system meant that USACO could hold competitions on the Internet, which led to an era of six annual contests (five shorter contests at three hours each; one five hours). These contests made it much easier to choose the best contestants to attend camp, since so many "performance data points" were collected. Open to the entire Internet community, it was not unusual for more than 1000 competitors from more than 65 countries to participate (several of whom were past their pre-college career but wanted the exposure and were ranked separately). A later upgrade enabled contests to be translated quickly, right before a contest began, thus expanding their scope.

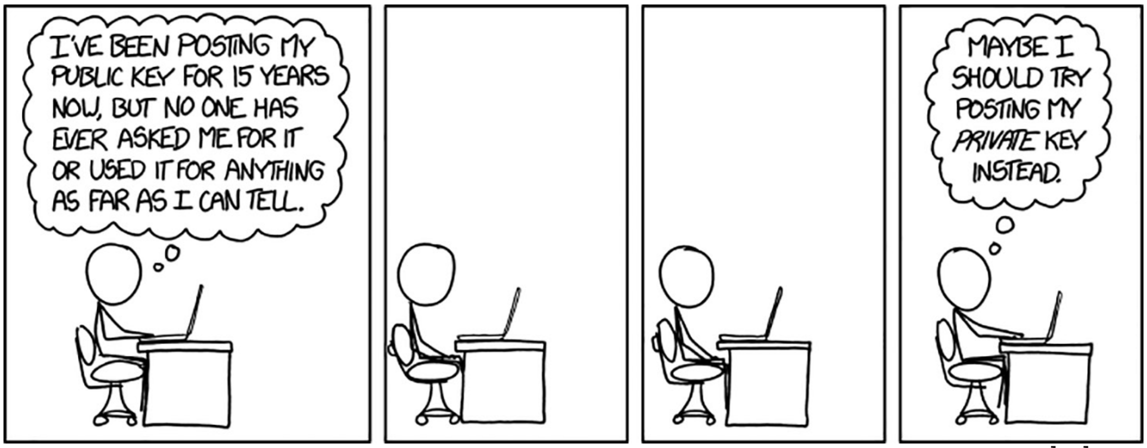
Once that challenge was met, it was obvious that an online training system could move the students more quickly through the complexities of "algorithmic programming." This engendered "The USACO Training" pages at http://train.usaco.org. Open to the entire Internet community, the site enforces a path through a curriculum, with 100 sample tasks that amplify the concepts of its tutorial sections. Almost a quarter million folks from around the world are registered for the training pages, which have also been translated into half a dozen languages for international students.

Each contest featured three tasks in each of three divisions, nine tasks total. Six contests required 54 tasks, each with 10–20 test data sets, validation, several sample solutions in various programming languages, and an analysis to be shared after the contest. The USACO camp featured an additional 5–6 contests (only one or two divisions) of three tasks in addition to a game-AI-challenge or other challenge—a total of 18–36 more tasks and the special software to run the challenge. This totals up to at least 72 tasks per year, repeated with new tasks each year.

Keeping track of these tasks became a challenge, so I built the contest-task management system based on some heuristics developed by Greg Galperin, Hal Burch, and others at camp. It keeps track of task text, coaches' solutions, test data, validator, validation status, the analysis, the difficulty (and other) ratings, and other miscellaneous information. Ultimately, this system could create a contest with three divisions and ensure the contest started on time and finished on time—automation at its finest. Of course, being data-driven, one had to ensure that the data was present, a huge task recurring monthly through the school year.

In summary, yeah, it's a lot of work to run the season. A set of a dozen coaches online throughout the year complements perhaps eight coaches (usually the same folks) invited to keep the camp (now with more than 30 participants) running smoothly. Two coaches and four students participate in the IOI and, occasionally, other international competitions. It's great fun and a real inspiration for those students who are interested. See http://www.usaco.org for more information.

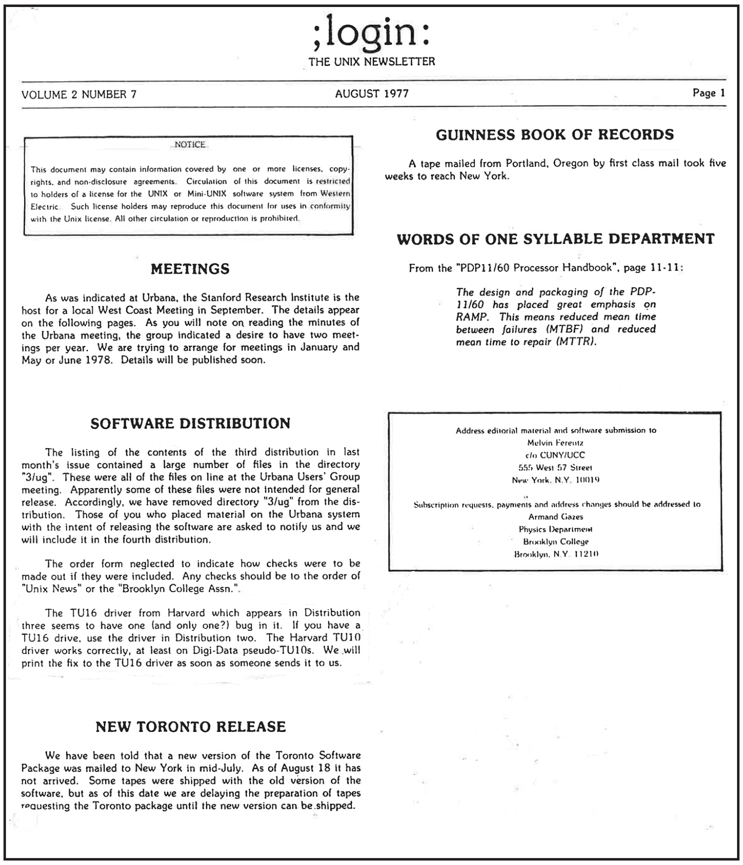
XKCD


I'VE BEEN POSTING MY PUBLIC KEY FOR 15 YEARS NOW, BUT NO ONE HAS EVER ASKED ME FOR IT OR USED IT FOR ANYTHING AS FAR AS I CAN TELL.
MAYBE I SHOULD TRY POSTING MY *PRIVATE* KEY INSTEAD.

xkcd.com

;login: The UNIX Newsletter

Volume 2, Number 7, August 1977



;login:

THE UNIX NEWSLETTER

VOLUME 2 NUMBER 7 AUGUST 1977 Page 1

NOTICE

This document may contain information covered by one or more licenses, copyrights, and non-disclosure agreements. Circulation of this document is restricted to holders of a license for the UNIX or Mini-UNIX software system from Western Electric. Such license holders may reproduce this document for uses in conformity with the Unix license. All other circulation or reproduction is prohibited.

MEETINGS

As was indicated at Urbana, the Stanford Research Institute is the host for a local West Coast Meeting in September. The details appear on the following pages. As you will note on reading the minutes of the Urbana meeting, the group indicated a desire to have two meetings per year. We are trying to arrange for meetings in January and May or June 1978. Details will be published soon.

SOFTWARE DISTRIBUTION

The listing of the contents of the third distribution in last month's issue contained a large number of files in the directory "3/ug". These were all of the files on line at the Urbana Users' Group meeting. Apparently some of these files were not intended for general release. Accordingly, we have removed directory "3/ug" from the distribution. Those of you who placed material on the Urbana system with the intent of releasing the software are asked to notify us and we will include it in the fourth distribution.

The order form neglected to indicate how checks were to be made out if they were included. Any checks should be to the order of "Unix News" or the "Brooklyn College Assn.".

The TU16 driver from Harvard which appears in Distribution three seems to have one (and only one?) bug in it. If you have a TU16 drive, use the driver in Distribution two. The Harvard TU10 driver works correctly, at least on Digi-Data pseudo-TU10s. We will print the fix to the TU16 driver as soon as someone sends it to us.

NEW TORONTO RELEASE

We have been told that a new version of the Toronto Software Package was mailed to New York in mid-July. As of August 18 it has not arrived. Some tapes were shipped with the old version of the software, but as of this date we are delaying the preparation of tapes requesting the Toronto package until the new version can be shipped.

GUINNESS BOOK OF RECORDS

A tape mailed from Portland, Oregon by first class mail took five weeks to reach New York.

WORDS OF ONE SYLLABLE DEPARTMENT

From the "PDP11/60 Processor Handbook", page 11-11:

The design and packaging of the PDP-11/60 has placed great emphasis on RAMP. This means reduced mean time between failures (MTBF) and reduced mean time to repair (MTTR).

Address editorial material and software submission to
Melvin Ferentz
c/o CUNY/UCC
555 West 57 Street
New York, N.Y. 10019

Subscription requests, payments and address changes should be addressed to
Armand Gazes
Physics Department
Brooklyn College
Brooklyn, N.Y. 11210

The second issue of *;login:* (formerly *UNIX News*) was published in August 1977 and included the minutes of the First National UNIX Users group meeting held May 19–21, 1977. Excerpts from that issue are reproduced here and on the following pages. We have also scanned the complete issue and made it available for download at www.usenix.org/login/dec15/login_aug77.pdf.

Minutes of the First National UNIX Users Group Meeting

Steven Zucker, *Interactive Systems Corporation*

The First National Meeting of the UNIX User's Group was held at the University of Illinois, Urbana-Champaign Campus, on May 19–21, 1977. Steve Holmgren of the University's Center for Advanced Computation chaired the meeting. The enthusiasm of the more than 150 participants and the informal tone of the sessions resulted in a very stimulating atmosphere for the exchange of ideas. The meeting was divided into eight sessions:

- UNIX Site Activities
- UCLA Data Secure UNIX
- Interprocess Communications
- Graphics
- Languages
- Networking
- Data Base Management Systems
- Phototypesetting

My hope is that these notes on the sessions will be useful in directing those wishing more details to people who can provide them. I offer my apologies to those whose contributions I have inadvertently omitted and urge them to send their contributions to this Newsletter.

Many of the sessions were replete with announcements by speakers as well as members of the audience of new and/or improved drivers for one or another device, with the T1-16 Magnetic Tape Unit receiving the most attention. Rather than list all the drivers mentioned here, I would like to suggest that a column in the UNIX NEWSLETTER be devoted to information of this kind with installations or individuals willing to disseminate such code supplying information as to features and requirements.

UNIX Site Activities

Heinz Lycklama, *Bell Telephone Laboratories*

Heinz described the several variants of UNIX that have been or are being developed at Bell Labs. In addition to the standard UNIX system which Western Electric already licenses, there are three other systems in use at Bell Labs.

LSI UNIX (LSX): LSX occupies 8K words of main memory leaving up to 20K words of user space for the single user that it supports. Minimum memory requirements for running LSX are 20K words of main memory, the extended instruction set and two floppy disks. LSX is written in C and will run the C compiler. It runs at most three processes and supports the notion of contiguous files but pipes are not supported.

LSX will run on the 11/10, 11/20, 11/34, or 11/40 as well as the LSI 11.

MINI-UNIX: Mini-UNIX supports up to four users running up to 13 concurrent processes on an 11/40, 11/34, 11/20, or 11/10. It occupies 12K words of memory leaving up to 16K words for user programs. It uses no memory mapping and, therefore, provides no memory protection. It requires an RK05 or larger disk.

MERT (Multi-Environment Real-Time System): This system runs only on an 11/70 or 11/45 as it requires the separation of kernel and supervisor spaces. MERT supports a real-time supervisor which can lock processes in memory, perform preemptive scheduling or time-out scheduling. The communication facilities (events, messages, shared memory, and process ports) were described. File system support for MERT is embodied in independent processes which communicate with other levels via messages.

* * * * *

Ken Thompson, *Bell Telephone Laboratories*

An effort is under way at Bell Labs to convert UNIX to run on the INTERDATA 8/32. The conversion is being treated primarily as a portability exercise. As part of the portability exercise a pseudo C has been developed which enforeces strict typing of variables.

A significant number of file system changes are being planned for Version 7 of UNIX. The changes would extend the allowable number of blocks in a file system from the present 2^{16} to 2^{24} blocks, thus, making it easier to use large disk files such as the RP04. The i-node size will be extended from the present 16 words to 32 words which will include space for 10 direct block pointers, one indirect block pointer, one double indirect pointer, and one triple indirect block pointer, allowing files to be as long as 2^{32} bytes. Users IDs will be extended to 16 bits and the STAT and the FSTAT system calls will hide the physical addresses. A long SEEK system call will replace the present SEEK and a TELL system call (the inverse of SEEK will be added). The SWITCHES call will be thrown away and the SLEEP call will be replaced by PAUSE and ALARM. Significant changes are also anticipated in the STTY and GTTY system calls. It is unlikely that the new system will be available before the beginning of 1978.

* * * * *

UCLA's Data Secure UNIX

Jerry Popek, *University of California, Los Angeles*

Jerry Popek described work at UCLA in which a secure version of the UNIX operating system is being developed. The system architecture is based on a kernel architecture, with program verification methods being applied to that software.

The kernel is composed of an operating system nucleus, smaller and simpler than the UNIX kernel, which is responsible for all operational security. It provides a "capability machine" with a number of simple kernel calls. Each one provides a primitive operating system function, such as process invocation, swapping, I/O, etc. Above the kernel, running in supervisor mode, is a "UNIX interface" module, that is part of each user's process (a process has two address spaces). That module is respinsible for providing an interface to user code that is identical to UNIX, and either performs the function, or prepares kernel requests to accomplish them if security relevant.

The secure UNIX system Popek described is to be capable of supporting large numbers of processes, and running virtually all non-super-user code without any change. A prototype implementation has been delivered to the government, and they are in the process of letting a contract to build a production version of secure UNIX.

Popek also described the program verification procedures necessary to show that protection is enforced by the system in an uncircumventable way.

* * * * *

R. M. Walden, *Western Electric*

Following the Data Secure UNIX presentation, Bob Walden announced that the Government and International Systems Division of Western Electric has established an organization to provide support for UNIX, initially to government users. The service will include consultation, installation and training, trouble shooting or problem solving assistance, improved documentation, and new feature development.

* * * * *

Interprocess Communication

Alan Nemeth, Bolt Beranek & Newman

Alan Nemeth reported on a series of meetings held to discuss interprocess communication in UNIX. The immediate goals of the meetings was to standardize on one or more interprocess communication mechanisms to be supported in UNIX systems run by the Department of Defense.

Two such mechanisms have been tentatively adopted: the port mechanism developed at The Rand Corporation and events developed at the University of Illinois. The Rand port mechanism described in Rand Report, R-2064/2, *Interprocess Communication Extensions for the UNIX Operating System,* provides a mechanism very much like pipes except that they can be named and opened by readers unrelated to the creator of the port. Ports also support message-oriented (as opposed to stream-oriented) I/O. While ports are intended for transfers of large amount of data between unrelated processes, the Illinois event mechanism provides a more efficient path for small one or two word messages. Each process has associated with it one event queue. Processes, using primitives provided by the kernel, can send "events" to ther processes, read an event from its queue if one is there, or wait for an event to appear on its queue. At present the development of a suitable signalling mechanism to augment ports and events and provide process synchronization is a subject for further study.

* * * * *

Following an open discussion of ports, events, and synchronization, there were two presentations of segment sharing mechanism that have been employed in UNIX systems. Heinz Lycklama of Bell Telephone Laboratories described the MERT interprocess communication facilities. MERT provides messges which are very similar to the Illinois events except that the messages may be somewhat longer—10 words instead of 2. The messages are employed in MERT for communication between the file manage process and the MERT kernel. In MERT the user is given the capability of manipulating memory segments. The user may have up to 32 segments—6 of which may be in his active address space.

Following Heinz's presentation, Steve Holmgren described the Illinois segment sharing mechanism by which processes may send segments to or receive segments from other processes.

* * * * *

Graphics

A special meeting of those attendees interested in graphics under UNIX was held on the evening of May 19. Karl Kelly of the University of Illinois Center for Advanced Computation presided at the meeting. This was a very informal session at which each manufacturer of graphics hardware took its knocks. The main conclusion that could be drawn from this session was that there exists a very large and very active group doing graphics under UNIX on a tremendous variety of equipment. It is probably safe to say that there is a UNIX driver for most of the common commercially available graphics devices.

* * * * *

Languages

Mike O'Brien, *University of Illinois*

Mike spoke very briefly about a new C compiler which supports long variables with initialization, structures, containing variables with byte fields, conditional compilation, structure initialization, and a new printf program.

* * * * *

Steve Bunch, *University of Illinois*

Steve spoke about a C compiler for the Honeywell level 6 which is to be in the public domain.

At this point in the meeting there were a number of announcements made from the floor of various languages available under UNIX from various sources. In particular Commercial Union Leasing Corporation, New York City, apparently has a C to FORTRAN processor as well as FORTRAN IV PLUS running under UNIX. Reports have indicated that the Commercial Union FORTRAN IV PLUS is vastly better than UNIX FORTRAN and it is available for a license fee from Commercial Union.

* * * * *

Tucker Taft, *Harvard University*

Tucker described ECL, an extensible language that is being run at Harvard. Documentation is available from the Harvard Science Center.

* * * * *

Arthur Olson, *University of California, San Diego*

Arthur Olson announced that San Diego was running the 11/40 floating point under UNIX.

* * * * *

Evelyn Walton, *University of California, Los Angeles*

Evelyn Walton made a brief presentation of the Pascal to C translator being used by the UCLA security kernel project. The purpose of the translator was to enable the production of code in Pascal for which an automatic verifier exists. No attempt was made to translate all of Pascal to C. Thus the translator does not support sets or nested procedures.

* * * * *

Peter Weiner, *Interactive Systems Corporation*

At this point in the meeting, Peter Weiner of Interactive Systems Corporation solicited suggestions from the floor on areas of UNIX that needed improvement or extension. Several euch suggestions were forthcoming especially in the networking area.

* * * * *

Networking

Jody Kravitz, *University of Illinois*
Steve Abraham, *University of California, Los Angeles*

The first networking presentation was made jointly by Jody Kravitz and Steve Abraham. They announced that there will be an official release of the UNIX ARPANET NCP (Network Control Program) combining changes made at the University of Illinois, at UCLA, and at Rand. The new release will be available in mid-summer from ILL-NTS.

* * * * *

Ken Thompson, *Bell Laboratories*

Ken described an experimental UNIX networking facility he is working on at Bell Laboratories. An interesting feature of the network is a protocol which provides a "directory assistance" facility. A demon process on each host accepts calls for directory assistance and provides routing information based on the part of the network that it knows about. The call initiation protocol establishes a path between the nodes on the network from source to destination and the messages transmitted from the source to the destination all follow the same path.

Present plans call for the use of a new DEC device, the KMC-11, a small microprocessor, to provide support for the network.

* * * * *

Brian Lucas, *National Bureau of Standards*

Brian Lucas discussed the ETHERNET. The ETHERNET provides a very high bandwidth yet low cost means of connecting machines that are within a single building or cluster of buildings. The network utilizes coaxial cables which support a one to two megabaud signalling rate. Adding a new host is as simple as connecting to the cable with a high impedance tap. Microprocessors between the host and the cable perform the actual signalling and detect and resolve conditions in which more than one host tries to signal simultaneously.

* * * * *

Data Base Management Systems

John Hoskins, *Office of Institutional Research*

John Hoskins described the Yale University Registration System. The system manages ten years of Yale undergraduate student records. Each student record is a separate file and the system holds approximately 15,000 records of two to three thousand bytes each. The system has been very well received in the Registrar's office where personnel are trained in only 4 hours and become expert in the use of the system in only two weeks. The primary components of the system are the Text Editor, a program called the "fence" (which makes available an editable copy of a student's record and prevents simultaneous update) and a number of shell files for producing grade reports, class schedules, and other reports as required. Those involved with it—both developers and users—speak very highly of the convenience and economy of using UNIX—even when compared with other larger and more expensive operating systems and machines.

* * * * *

Dan Giclan, *New York Telephone*

Dan Giclan reported on the development of an enhanced, production version of the INGRES system developed at Berkeley. The improved system is oriented towards production use rather than theoretical completeness. By vesting ownership of the data bases in the user rather than the system and by placing responsibility for avoiding the rare but potentially dangerous problem of concurrency (simultaneous update) the system is able to run ten times faster than the original Berkeley system.

* * * * *

Bill Mayhew, *The Children's Museum*

Bill Mayhew described "The Information System," available for license [from the Children's Museum].

The Information System is distributed as a collection of routines that perform the standard database operations: add item, delete item, add descriptor to item, remove descriptor from item, locate descriptor, retrieve next item in inverted list, delete descriptor from dictionary, plus the AND, OR, and AND NOT hitlist boolean operators. Also supplied is a user interface implementing a simple query language and providing facilities for entering, updating, and retrieving textual data items.

The Information System can be applied to a wide range of information management problems. It has been successfully used to develop interactive maintenance systems for mailing lists, membership and contribution records, and group visit and educational program reservations, and is about to be used as the foundation for a service to match the educational resources of cultural organizations with the needs of teachers throughout the Commonwealth of Massachusetts.

* * * * *

Phototypesetting

Joe Ossana, *Bell Laboratories*

The principal speaker at the Phototypesetting session was Joe Ossana. He announced a new phototypesetting package, typesetter V7, which is or soon will be available from Western Electric for a $3300 license fee. The new package combines NROFF and TROFF and is written in C. This results in TROFF being 50% larger and 20–30% slower than the earlier version. There have been a number of significant improvements in the package however. First TROFF font control and width calculations are now taken from files so it is relatively easy to use the package to drive other phototypesetters. Second, one can now specify artificial bolding which is performed by overstriking characters with a small offset. EQN the mathematics typesetting program now works with NROFF.

Also:

- Bell Labs is looking into other typesetters (an APS4 or APS5 typesetters, which sells for about $100,000). TROFF will easily drive it, although making use of the advanced features such as more fonts or sizes may be difficult.
- Bell Labs has a Tektronics 4014 TROFF simulator which, though slow, can show what a typeset page will look like.
- Measurements have indicated that NROFF hypenation is correct approximately 97-1/2% of the time.
- There is a new columnar cable builder

* * * * *

Two other announcements were made at the Phototypesetting session.

Larry Smith, *Texas Student Publications*

Larry announced that the University of Texas at Austin Student Publications are running two PHOTON typesetters under UNIX.

* * * * *

Gerry Barksdale, *Naval Postgraduate School*

Gerry announced the availability of fonts which can be printed on a VERSATEK printer.

* * * * *

During a break in the Phototypesetting session, two awards were presented. The first was presented by Greg Chesson to Steve Holmgren for his work in organizing and chairing the conference. Steve was pleased to receive a stuffed pheasant for his mantlepiece. The second award was presented by Ken Thompson to Dennis Mumaugh of the Department of Defense for having the largest collection of UNIX users software in existence.

Anyone wishing to purchase an angry-looking rubber chicken should get in touch with Dennis.

* * * * *

Users Group Business

Mel Ferentz, *City University of New York*

Mel announced that the UNIX Users Group will be incorporating as a nonprofit educational organization in order to obtain such benefits as favorable postage rates on its mailings. He also announced that the software distribution center will be moving from Chicago Circle to New York where the availability of greater machine resources will make it possible to speed the delivery of new distributions. The schedule for the next Users Group meeting was discussed; it will be published in the NEWSLETTER when fixed.

* * * * *

COLUMNS

Awaiting for Godot

DAVID BEAZLEY

David Beazley is an open source developer and author of the *Python Essential Reference* (4th Edition, Addison-Wesley, 2009). He is also known as the creator of Swig (http://www.swig.org) and Python Lex-Yacc (http://www.dabeaz.com/ply.html). Beazley is based in Chicago, where he also teaches a variety of Python courses.
dave@dabeaz.com

Python 3.5 was released to the world on September 13, 2015. Included in this release was a substantial upgrade to asynchronous I/O support in the `asyncio` module, including brand new syntax in the Python language itself [1]. In this article, we dive into modern `asyncio` and take it for a test drive on code involving low-level socket programming, high-level sockets, HTTP clients, and HTTP servers. Prepare yourself for the unexpected—this is not the Python you know. Just to be clear, you'll need Python 3.5 or newer to try the examples.

But First, Some Beckett

To be honest, I've never considered myself to be much of a theater nut. In fact, my wife often teases me about how easily I fall asleep at shows. However, much to her chagrin, I find myself to be a huge fan of Samuel Beckett plays. For reasons that will become clear shortly, the title of this article is a reference to Beckett's most famous work, "Waiting for Godot," a play in which nothing actually seems to happen! I like such plays—they're easier to follow.

Beckett plays are not your ordinary kind of theatrical affair. For example, a few years back, I found myself riveted during a production of *Krapp's Last Tape*, a play where not a single word is spoken for the first 25 minutes. Instead the main character slowly paces around stage eating a banana—deep in thought. What is this? What is going on here? Or as a more extreme example, there is Beckett's *Play*—a production in which three motionless heads, situated atop funeral urns, talk frenetically amongst themselves at such a rapid pace, you can't make any sense of what is actually happening or what it is about. The play suddenly ends, the stage lights go out, and you're left wondering, "WHAT was THAT?" No, wait, the lights come back on and they simply repeat the whole play word-for-word start-to-finish again. What? As I said, Beckett plays defy convention.

This brings me to the topic of this article—asynchronous I/O and Python's new `asyncio` module. Like a Beckett play, `asyncio` defies normal Python conventions. At times, I don't know what exactly I'm looking at, and I'm not even sure if I like it. However, it's never boring. If anything, understanding `asyncio` will push your Python knowledge to the very edge of what's possible. First added in Python 3.4, `asyncio`'s goal was to reboot Python's support for asynchronous I/O and to take advantage of programming techniques involving advanced use of generator functions. However, in Python 3.5, it has taken flight with new language syntax and constructs. To the uninitiated, the code looks foreign and crazy. Is this even Python? What is this? What is going on?! As a whole, the Python community tends toward the conservative (just consider the number of people still using Python 2). Yet Python 3.5 might represent the most radical departure from the ordinary that I can recall in my nearly 20 years of Python coding. Yes, it's that different.

Although `asyncio` has been brewing in Python for a few years, I've never really quite figured out how to tackle it as a topic. Should I dive into its history and talk about the internals that make it work? Or should I just throw people into the deep end of the pool and see what happens? This article takes the latter approach. Assuming you have never done any prior async programming in Python and know nothing about past efforts, what does it look like in Python 3.5? This is what we're going to find out. Part of the exercise is to see if using `asyncio` can be as easy and useful as more traditional parts of Python.

Some Background

Before beginning, it's useful to know why you might consider asynchronous I/O. In many modern network applications, it is common to have to a huge number of connected clients. These clients typically don't involve tons of high-bandwidth I/O—it's just that there are a lot of them (imagine a single server process with 10,000–20,000 open clients). Classic techniques for managing concurrency such as threads and processes have known limitations working with such a large number of clients. Asynchronous I/O takes a different approach. Typically, a single process runs an event-loop that polls for activity on all of the clients and handles it in some way, such as triggering event callback functions. However, callbacks have their own problem—leading to what's commonly referred to as a kind of "callback hell" of spaghetti code.

The `asyncio` module takes a different approach involving coroutines. Coroutines look a lot like normal synchronous code but are driven by an event loop under the covers. This tends to lead to code that is much easier to write and reason about. However, coroutines are not the same as ordinary functions or procedures. Thus, Python's `asyncio` module has a rather different flavor than the rest of the standard library.

To explore `asyncio`, we will look at four common problems involving network programming: programming directly with sockets, creating high-level socket servers, interacting with HTTP services, and creating a simple HTTP server. You'll probably want to hang on for the ride. Let's begin.

Low-Level Socket Programming

One of my first uses of Python involved writing simple network services directly using sockets [2]. Having previously written such code in C, it was an enlightening experience to see how easy it was in Python. For example, here is a simple multithreaded echo-server:

```
# A simple echo server using sockets and threads

from socket import *
from threading import Thread

def echo_server(address):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    sock.bind(address)
    sock.listen(5)
    while True:
        client, addr = sock.accept()
        print('Connection from:', addr)
        Thread(target=echo_client, args=(client,),
            daemon=True).start()

def echo_client(client):
    while True:
        data = client.recv(1000)
        if not data:
            break
        client.sendall(data)
    print('Connection closed')
    client.close()

if __name__ == '__main__':
    echo_server(('', 25000))
```

It's simple, yet perhaps a bit dicey with the potential for unlimited thread creation as the number of clients increases.

As a first test of `asyncio`, let's see if we can write the same code without much fuss. Here is a direct translation of the code to an asynchronous version:

```
# A simple server using asyncio and sockets

from socket import *
import asyncio

async def echo_server(loop, address):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    sock.bind(address)
    sock.listen(5)
    sock.setblocking(False)   # Critical
    while True:
        client, addr = await loop.sock_accept(sock)
        print('Connection from:', addr)
        task = loop.create_task(echo_client(loop, client))

async def echo_client(loop, client):
    while True:
        data = await loop.sock_recv(client, 1000)
        if not data:
            break
        await loop.sock_sendall(client, data)
    print('Connection closed')
    client.close()

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(echo_server(loop, ('', 25000)))
```

At first glance, this code will shatter your mind. `async def` and `await`? What is this business? It's new Python syntax related to asynchronous programming. The `async def` statement declares a function as a coroutine that must be managed by an event loop. The `await` statement is used to execute a coroutine and return its result. The two statements are meant to work together.

As noted, the execution of the code requires the use of an underlying event loop. The `asyncio.get_event_loop()` and `loop.run_until_complete()` calls at the end of the program show how you obtain a reference to the loop and initiate execution of the program.

Other operations in the code have changed into methods involving the event loop. For example, the `loop.sock_recv()`, `loop.sock_accept()`, and `loop.sock_sendall()` carry out the standard socket operations. The `loop.create_task()` method launches a new task much like the creation of a thread.

If you run the code, you'll find that it works just as it did before. Concurrent connections also work fine even though no threads or subprocesses are involved. Great!

High-Level Sockets

For most, directly programming with sockets is not the most ideal way to write network applications. Python has long provided a higher-level interface in the `socketserver` module (named `SocketServer` in legacy Python) [3]. Here is another implementation of an echo server using the streams interface of `socketserver`:

```
# Echo server using socketserver

from socketserver import (
    StreamRequestHandler,
    ThreadingTCPServer
)

class EchoHandler(StreamRequestHandler):
    def handle(self):
        print('Connection from:', self.client_address)
        for line in self.rfile:
            self.wfile.write(line)
        print('Connection closed')

if __name__ == '__main__':
    serv = ThreadingTCPServer(('', 25000), EchoHandler)
    serv.serve_forever()
```

In this code, the `EchoHandler` class has `rfile` and `wfile` attributes, which operate like files for reading and writing network data. A simple `for`-loop can be used to read line-by-line as shown. This loop will terminate when the connection is closed.

As a second test of `asyncio`, can we write a similar high-level program? Here is the answer:

```
# Echo server using asyncio and streams

import asyncio

async def handle_echo(reader, writer):
    print('Connection from:', writer.get_extra_info('peername'))
    async for line in reader:
        writer.write(line)
        await writer.drain()
    print('Connection closed')
    writer.close()

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    coro = asyncio.start_server(handle_echo, '', 25000, loop=loop)
    server = loop.run_until_complete(coro)
    loop.run_forever()
```

Although the code is packaged in a slightly different way (no need for a class definition), it is comparably short and similar in logic. You'll find that it handles concurrent client connections even though no threads are being used.

Again, `async def` is being used to declare the handler as coroutine. The `await writer.drain()` statement waits for the written data to be successfully sent. A new mystery arises in the `async for` statement. What in the world is that? In short, it's a for-loop whose iteration requires coordination with the event loop (e.g., the event loop has to suspend the code and resume it when data actually arrives). If you don't use `async for`, the code would change slightly to the following:

```
async def handle_echo(reader, writer):
    print('Connection from:', writer.get_extra_info('peername'))
    while True:
        line = await reader.readline()
        if not line:
            break
        writer.write(line)
        await writer.drain()
    print('Connection closed')
    writer.close()
```

That's not nearly as elegant. Besides, dropping some Python code with an `async for` loop in it on your coworkers will be a good thing for them. Do it.

Making HTTP Requests

One of my favorite libraries for interacting with the Web is the `requests` library [4]. Using it is easy:

```
>>> import requests
>>> r = requests.get('http://www.python.org')
>>> print(r.text)
... see returned result ...
```

It's almost too easy. Of course, a really good request needs some threads. And threads need semaphores. And why not a queue for good measure? Behold:

```
# get.py
#
# Fetch data from a collection of URLs using requests and threads

import requests
from queue import Queue
from threading import Thread, Semaphore

max_active = Semaphore(4)

def url_worker(url, result_queue):
    with max_active:
        r = requests.get(url)
        result_queue.put((url, r.status_code, r.text))

def get_urls(urls):
    result_queue = Queue()

    # Launch workers
    workers = []
    for url in urls:
        worker = Thread(target=url_worker, args=(url,
                result_queue))
        worker.start()
        workers.append(worker)

    # Wait from the workers to finish and collect results
    results = { }
    for worker in workers:
        worker.join()
        url, status, text = result_queue.get()
        results[url] = (status, text)

    return results

if __name__ == '__main__':
    urls = [
        'https://docs.python.org/3/library/asyncio.html',
        'https://docs.python.org/3/library/select.html',
        'https://docs.python.org/3/library/threading.html',
        'https://docs.python.org/3/library/selectors.html',
        'https://docs.python.org/3/library/queue.html',
        'https://docs.python.org/3/library/socket.html',
        'https://docs.python.org/3/library/socketserver.html',
        ]
    result = get_urls(urls)
```

In this program, the `get_urls()` function takes a list of URLs and spawns a collection of worker threads to fetch data concurrently. The workers are throttled using a semaphore. Results are returned via a queue. The final result is a dict mapping URLs to a status code and text from the response. It's a program that only a parent process could love. Take that async!

A similar program can be written for asyncio using the third-party aiohttp library [5]. Using that, here's an async version:

```
# aget.py
#
# Fetch data from a collection of URLs using
# aiohttp and asyncio

import aiohttp
import asyncio

max_active = asyncio.Semaphore(4)

async def url_worker(url, result_queue):
    async with max_active:
        r = await aiohttp.get(url)
        await result_queue.put((url, r.status,
               await r.text()))

async def _get_urls_task(urls, loop):
    result_queue = asyncio.Queue()

    # Launch workers
    workers = []
    for url in urls:
        worker = loop.create_task(url_worker(url,
                result_queue))
        workers.append(worker)

    # Wait for the workers to finish
    results = { }
    for worker in workers:
        await asyncio.wait_for(worker, timeout=None)
        url, status, text = await result_queue.get()
        results[url] = (status, text)

    return results

def get_urls(urls):
    loop = asyncio.get_event_loop()
    result = loop.run_until_complete(_get_urls_task(urls, loop))
    return result
```

```
if __name__ == '__main__':
    urls = [
        'https://docs.python.org/3/library/asyncio.html',
        'https://docs.python.org/3/library/select.html',
        'https://docs.python.org/3/library/threading.html',
        'https://docs.python.org/3/library/selectors.html',
        'https://docs.python.org/3/library/queue.html',
        'https://docs.python.org/3/library/socket.html',
        'https://docs.python.org/3/library/socketserver.html',
        ]

    result = get_urls(urls)
```

Does it work? You bet, but there are a variety of details. First, the aiohttp module provides its own get() method for making an HTTP request. It's similar to the requests library except that you need to use r = await aiohttp.get() when making the request and await r.text() to obtain the downloaded data.

Next, the asyncio module provides its own version of synchronization primitives and queues. So the code uses asyncio.Semaphore and asyncio.Queue. The async with max_active statement in url_worker performs an asynchronous acquisition of the associated semaphore. All queueing operations are prefaced by the await keyword to indicate their asynchronous nature. The asyncio.wait_for() call waits for a task to finish. In many ways, it's not too dissimilar from threads. One subtle aspect concerns the separate _get_urls_task() and get_urls() functions. With asyncio nothing happens unless someone drives the underlying event loop. Thus, the get_urls() function works by setting up the calculation and driving the loop until the result comes back.

Stepping back for a moment, the only purpose of the queue in the threading example is to deal with the fact that there is no clean way to communicate results back from worker threads. It turns out you can eliminate it from the asynchronous code entirely. Here's a slightly modified version that is a bit cleaner:

```
import aiohttp
import asyncio

max_active = asyncio.Semaphore(4)

async def url_worker(url):
    async with max_active:
        r = await aiohttp.get(url)
        return url, r.status, await r.text()

async def _get_urls_task(urls, loop):
    # Launch workers
    workers = []
    for url in urls:
        worker = loop.create_task(url_worker(url))
        workers.append(worker)

    # Wait for the workers to finish
    results = { }
    for worker in workers:
        url, status, text = await asyncio.wait_for(worker,
                    timeout=None)
        results[url] = (status, text)
        print(url, status)

    return results
```

asyncio provides a fairly complete set of primitives normally associated with thread programming. These include locks, semaphores, events, queues, and more. Although these objects do not provide 100% compatibility with their threading counterparts, it seems that many programs written for threads could probably be adapted to asyncio in a straightforward way.

Writing a Web Service

As a final test, let's write a simple Web service. In this example, the third-party bottle library is used to implement an endpoint that computes Fibonacci numbers, returning the result as JSON [6]:

```
# Simple web service using bottle

import bottle

def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

@bottle.route('/fib/<n>;')
def serv_fib(n):
    result = fib(int(n))
    return { 'result': result }

if __name__ == '__main__':
    bottle.run(host='', port=25000, debug=True)
```

Connect to the service using a URL such as http://localhost:25000/fib/6 on your machine. Replace n by an integer number such as 6. You should get a response such as this:

```
{"result": 8}
```

Can this code be replaced by an asynchronous version? aiohttp to the rescue!

```
# Simple web service using aiohttp

from aiohttp import web
import asyncio
import json
```

```
def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

async def fib_handler(request):
    n = int(request.match_info['n'])
    result = fib(n)
    resp = { 'result': result }
    return web.Response(text=json.dumps(resp),
             content_type='application/json')

def run(address):
    loop = asyncio.get_event_loop()
    app = web.Application(loop=loop)
    app.router.add_route('GET', '/fib/{n}', fib_handler)
    srv = loop.create_server(app.make_handler(), *address)
    loop.run_until_complete(srv)
    loop.run_forever()

if __name__ == '__main__':
    run(('', 25000))
```

Alas, it's not quite as compact as the `bottle` version, but it performs the same function.

One downside of this implementation is that the implementation of `fib()` is especially bad for large integers. Try giving a number such as 45 as input and watch how the whole server locks up (you can't make requests from other clients). This is the nature of async—long-running calculations will block the entire event loop. However, you can move the work out to a thread pool by using the `loop.run_in_executor()` function as follows:

```
from concurrent.futures import ThreadPoolExecutor
_pool = ThreadPoolExecutor()

async def fib_handler(request):
    n = int(request.match_info['n'])
    if n < 15:
        result = fib(n)
    else:
        loop = asyncio.get_event_loop()
        result = await loop.run_in_executor(_pool, fib, n)
    resp = { 'result': result }
    return web.Response(text=json.dumps(resp),
             content_type='application/json')
```

If you try this modified version with a large integer, you'll find that the computation no longer blocks everything—you'll be able to make other requests while the calculation is churning away. Excellent. A similar technique would need to be used for any code that might potentially block the event loop.

Thoughts

In this article, I've presented a few examples of using `asyncio` with variations of the modern `async` syntax. At first glance, the code is probably a bit jarring—it looks unlike any Python code you might have written before. Yet, at the same time, the code retains the clarity of synchronous code written to use threads or processes. That is certainly a big plus.

A major concern with `asyncio` is that it tends to be an "all in" prospect for software development. If you're going to use it, you need to make sure that all of your libraries and code support it. Particular attention needs to be given to blocking operations involving files, network connections, subprocesses, and other I/O related tasks. A big part of the new "async" and "await" syntax is making the distinction between synchronous and asynchronous code absolutely clear. As a general rule, any existing Python code will probably run in an asynchronous environment, but unless "await" is being used, there's no guarantee that it won't block the event loop by accident. Whenever possible, you'll want to use libraries that have been written with `asyncio` in mind.

On that subject, just what libraries are available? At this time, there seems to be a growing set of modules for interacting with a variety of network services. As noted, `aiohttp` provides async support for HTTP, both as a client and a server. The library provides additional support for WebSockets and other modern HTTP features. A search for "asyncio" on the Python Package Index reveals a wide variety of libraries for interacting with services such as Redis, ZeroMQ, MondoDB, XML-RPC, IRC, Postgres, and others. Naturally, many of these are new and experimental. However, it seems that Python's asynchronous future is poised to be rather interesting.

Finally, a few words of caution. In writing this article, I debated how much information to provide on how `asyncio` actually works under the covers. If anything, you really don't want to know how it works—if you go wandering into the source code, your mind will completely shatter into a million pieces. Let's just say that it involves a lot of very clever programming with generators. I wrote about some of the general concepts in a previous *;login:* article although `asyncio` takes it to a whole new level of complexity [7]. Also, if you're going to use `asyncio` you must heed every word of advice you can find in the documentation. If you decide to play it fast and loose, you're likely going to spend hours debugging and cursing. For example, I wasted the better part of a half-day trying to figure out why I couldn't get my simple socket example to work. As it turns out I forgot to put the socket in non-blocking mode—a detail found in the documentation, but which I overlooked at the time. The bottom line: you need to read the documentation carefully.

More Information

The documentation for `asyncio` should be your first starting point. PEP 3156 is a must read for background information and rationale for `asyncio` [8]. PEP 492 contains information on the new `async` and `await` syntax [9]. There have been numerous talks given about `asyncio` over the last year. There are too many to list individually, but a search of pyvideo will not disappoint [10].

References

[1] asyncio module: https://docs.python.org/3/library/asyncio.html.

[2] socket module: https://docs.python.org/3/library/socket.html.

[3] socketserver module: https://docs.python.org/3/library/socketserver.html.

[4] requests module: http://www.python-requests.org/en/latest/.

[5] aiohttp module: http://aiohttp.readthedocs.org.

[6] bottle module: http://bottlepy.org.

[7] David Beazley, "A Tale of Two Concurrencies (Part 2)," *;login:*, vol. 40, no. 4, August 2015: https://www.usenix.org/publications/login/aug15/beazley.

[8] PEP 3156: https://www.python.org/dev/peps/pep-3156/.

[9] PEP 492: https://www.python.org/dev/peps/pep-0492/.

[10] pyvideo: http://pyvideo.org.

Practical Perl Tools

OAuth2 in Situ

DAVID N. BLANK-EDELMAN

David Blank-Edelman is the Technical Evangelist at Apcera (the comments/views here are David's alone and do not represent Apcera/Ericsson). He has spent close to 30 years in the system administration/DevOps/SRE field in large multiplatform environments, including Brandeis University, Cambridge Technology Group, MIT Media Laboratory, and Northeastern University. He is the author of the O'Reilly Otter book *Automating System Administration with Perl* and is a frequent invited speaker/organizer for conferences in the field. David is honored to serve on the USENIX Board of Directors. He prefers to pronounce Evangelist with a hard "g."
dnblankedelman@gmail.com

In the last column I threw a little hissy fit around the authentication options for working with the WordPress REST API. In it, I made noises about the adoption of OAuth 2.0 over version 1.0a and further grumbled about the backwards incompatibility (not to mention some of the politics around the changes) between the two versions. All of this piqued the interest of my editor who asked me to write some more on the topic. I'm still not thrilled about the OAuth situation, but I thought I would try to redeem myself by providing a column around the subject based on an actual piece of code that had to authenticate using OAuth2. This still won't address the OAuth 1.0a questions, but perhaps future columns will drag me kicking and screaming in that direction as well. One brief aside about version 1.0a because I need to make a slight correction: in the previous column I had suggested that 2.0 had all but supplanted 1.0a in the world. I've recently been discovering a few pockets of 1.0a (for example, Twitter's API, probably for historical reasons, seems to consist of this strange mishmash of the two), so I don't think 1.0a can be considered dead quite yet. Maybe we'll make with the Twitter in a future column.

Background

For those of you who haven't had the pleasure of diving into either of the OAuth versions, let's take a quick moment to describe the beast and what purpose it serves. In many of the cases where you will initially encounter it, it will be used as an authentication protocol (albeit one that appears to be more complex than it needs to be). But OAuth was designed to be much more than that. Here's the very best description [1] of the intent I have ever seen, from a blog post by Eran Hammer, one of the former lead authors of the spec, which is also quoted on oauth.net, the canonical home for OAuth material:

> Many luxury cars today come with a valet key. It is a special key you give the parking attendant and unlike your regular key, will not allow the car to drive more than a mile or two. Some valet keys will not open the trunk, while others will block access to your onboard cell phone address book. Regardless of what restrictions the valet key imposes, the idea is very clever. You give someone limited access to your car with a special key, while using your regular key to unlock everything.
>
> Every day new websites launch offering services which tie together functionality from other sites. A photo lab printing your online photos, a social network using your address book to look for friends, and APIs to build your own desktop application version of a popular site. These are all great services—what is not so great about some of the implementations is their request for your username and password to the other site. When you agree to share your secret credentials, not only do you expose your password to someone else (yes, that same password you

> also use for online banking), you also give them full access to do as they wish. They can do anything they wanted—even change your password and lock you out.
>
> This is the problem OAuth solves. It allows you, the User, to grant access to your private resources on one site (which is called the Service Provider), to another site (called Consumer, not to be confused with you, the User). While OpenID is all about using a single identity to sign into many sites, OAuth is about giving access to your stuff without sharing your identity at all (or its secret parts).

As an aside, in the process that led to 2.0, Hammer subsequently left the OAuth working group and withdrew his name from the specification because he thought 2.0 was deeply flawed. See his subsequent post, "OAuth 2.0 and the Road to Hell," [2] for more details. He's also got an entertaining, albeit NSFW, talk on the flaws of OAuth and the OAuth spec process [3]. I told you that OAuth was politically messy...

So a key thing to note about the OAuth intention statement above (that is brought out nicely in the first chapter of the upcoming book *OAuth2 in Action* by Justin Richer and Antonio Sanso) is that OAuth is less of an authentication protocol and more of a delegation protocol. The idea is that it can provide you a way to say, "Let this program/service/etc. have the access to do the following things on my behalf." If you've ever signed into a new mail or Twitter client and found yourself logging first into Gmail or Twitter to be faced with a screen of permissions to grant, you've likely been part of an OAuth transaction (of some version or other).

The Goal

Now that you know what sort of protocol we are dealing with, let's look at the actual goal of the script we're going to write together. Let's just say, hypothetically, that you work with an organization that uses Google Calendar to maintain its "in-out" listing (i.e., who is going to be out of the office, who is working from home, who is on vacation, etc.). Each person who is not going to be in the office marks this by making an event in this shared calendar. It can be a bit cluttered, but on the whole it works fine. There's just one niggling problem: sometimes people add calendar entries for their absences but forget to turn off notifications for those entries. This means that every day, everyone in the organization who subscribes to this calendar receives notifications (whether they like it or not) for those entries. We're going to write some code that connects to the Google calendar service, reads that calendar, and displays the events which have notifications still set. Then presumably we can go visit either the entries or the individuals in question and take corrective action.

Where does OAuth2 come in? Google forces you to use/interact with their (respected, even by Hammer) OAuth2.0 implementation to gain access to private calendars. Let's dive in.

```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Figure 1: OAuth2 protocol flow, from https://tools.ietf.org/html/rfc6749

How Does It Work?

Before we get into looking at actual code, we probably should take at least a few steps up the OAuth2 learning curve (a fairly steep slope, I might add). Even if some magic Perl module will take care of all of the details behind the scenes, it is really important to get at least a general sense of what is going on. To do that, I'm going to elide a whole bunch o' details because OAuth2 has a number of different "flows" whereby different steps get taken depending on just what context it is being used in (e.g., a Web application, some application running on your computer, an application running in the browser, etc.) and just what kind of access the person interacting with the system has to a real Web browser for one of the steps (e.g., is it being used from a machine with a browser? in some device with no keyboard? etc.). In this column, we're only going to show one flow/scenario that works for the task at hand. If you plan to get deeper into this stuff, I highly recommend you read and reread and reread the documentation of the vendor that you will be talking OAuth2 to/with. Some (e.g., Google [4]) have quite decent documentation, so perhaps you will get lucky.

Okay, let's toss a stone and skim the surface. The basic OAuth2 protocol flow looks like the diagram from RFC6749 in Figure 1.

The dance goes roughly like this: you (the client) send an "I'd like my app to have the following access for this service" request to the resource holder of your choice (Google, in our case). That resource holder does something to verify you and to obtain your consent to delegate this access to your app. It hands back an authorization token that says, "Yes, I've confirmed that this person is fine and allowed to access X." Your app can then send this token to an authorization server, which hands you back another token (an access token) that will act as a key to unlock access to the protected resource in question when presented at the same time as a request for that resource.

In the case of the code we are going to write, the script will send a request to Google's auth service requesting read-only access to an account's private calendars. Google will reply with a URL,

```
+--------+                                           +---------------+
|        |--(A)------- Authorization Grant --------->|               |
|        |                                           |               |
|        |<-(B)----------- Access Token -------------|               |
|        |               & Refresh Token             |               |
|        |                                           |               |
|        |                            +----------+   |               |
|        |--(C)---- Access Token ---->|          |   |               |
|        |                            |          |   |               |
|        |<-(D)- Protected Resource --| Resource |   | Authorization |
| Client |                            |  Server  |   |     Server    |
|        |--(E)---- Access Token ---->|          |   |               |
|        |                            |          |   |               |
|        |<-(F)- Invalid Token Error -|          |   |               |
|        |                            +----------+   |               |
|        |                                           |               |
|        |--(G)----------- Refresh Token ----------->|               |
|        |                                           |               |
|        |<-(H)----------- Access Token -------------|               |
+--------+           & Optional Refresh Token        +---------------+
```

Figure 2: Refreshing an expired access token, from https://tools.ietf.org/html/rfc6749

which we'll go to "by hand." That URL will take us through the standard Google login procedure (if not logged in) and then pop up a screen that will say, "{App Name} would like to view your calendars [Deny] [Allow]." If you click the Allow button, a subsequent screen will be presented that says, "Please copy this code, switch to your application and paste it there: {longish_string_of_random_characters}." That string is the authorization token. We'll type it into a prompt presented by the script, the script will exchange it for an access token (which it will hold on to for us), and finally it will begin to send requests to the Google Calendar API with that token added to the headers of all of the requests. We'll then parse the returned data looking for specific calendar entries.

That's how we're going to dance. But I would be a little remiss if I didn't mention one complication that we won't touch out of either brevity or laziness (you decide). To improve the security of the situation, access tokens all have relatively short expiration times after which they can no longer be used. The authorization token we originally received is only good for one access token exchange, so what do we do after the access token expires? OAuth2 brings yet another token into the picture (yes, that's three so far, but who's counting?) called a refresh token. Figure 2 depicts the dance taken to refresh a token.

This crazy little ASCII diagram shows that we can also request a refresh token as part of the dance. The refresh token is supposed to be squirreled away someplace safe, only be brought out and sent over the wire when it is time to get a new access token issued as expiration approaches/arrives. Our script isn't going to run long enough to hit expiration timeouts, so we're not going to bother with this token, nor are we going to pay attention to storing tokens, something you would do for more complex/longer-lived applications.

Code Time

With that background in mind, let's look at two code samples. To get warmed up, let's look at how you would get a list of the calendars I have configured in Google calendar (the list of calendars I own and have subscribed to on the left side of the Google Calendar Web app).

Now, it would be entirely possible to write all of the OAuth2 code using bare-bones Perl modules used for making HTTP/S requests like LWP::UserAgent or HTTP::Tiny. We've seen a number of these in the past. But in this case, I'm perfectly happy to let someone else work out some of the fiddly bits because we are bound to hit plenty of fiddly when we actually get to use the Calendar API.

To make my life a bit easier, I am using LWP::Authen::OAuth2. Think of it as LWP::UserAgent with some magic OAuth2 pixie dust mixed in (plus it has some Google-centric code baked in). At the very least, I'd encourage you to look at its Overview doc, which does a good job of writing up some of the background/issues you are sure to want to know about before diving into this stuff.

So we start like this:

```
use LWP::Authen::OAuth2;
use Browser::Open qw( open_browser );
use IO::Prompt::Tiny qw/prompt/;
use JSON;

my $cal_uri = 'https://www.googleapis.com/calendar/v3';

my $oauth2 = LWP::Authen::OAuth2->new(
   client_id =>
'getyourowncode.apps.googleusercontent.com',
   client_secret     => 'need your own secret',
   service_provider  => "Google",
   scope             => 'https://www.googleapis.com/auth/
                          calendar.readonly',

   redirect_uri => "urn:ietf:wg:oauth:2.0:oob",

   # Optional hook, but recommended.
   #save_tokens => \&save_tokens,
   #save_tokens_args => [ $dbh ],

   # This is for when you have tokens from last time.
   #token_string => $token_string.
);
```

After setting a variable for later use that shows the base URL for the Google Calendar API, we create a new oauth2 object. It requires a client_id and client_secret from Google. To get one of these, you will want to go to https://console.developers.google.com, create a new project, enable just the Google Calendar API for it, then under "Credentials" request a new OAuth2 client_id. When it asked for an application type, I selected “Other,” and that’s worked swimmingly for me for this script.

There are two other key parameters here. The first is “scope,” which is the level of access we are requesting to the resource. With Google Calendar, the choices are few (read-only or read-write access). The other parameter is the redirect_uri. This is the URI that will be handling the authorization token once it is issued. That little weird-looking line just says, “Display the token in the browser and ask the user to do something with it” (vs. going to a real URI). I left the token-related parameters from the documentation commented out just as a reminder that better token handling would normally be inserted at this point.

oauth2 object in hand, we then do this:

```
my $url = $oauth2->authorization_url();

open_browser($url);

my $auth_code = prompt("Please enter auth code
                       provided by Google:");

$oauth2->request_tokens( code => $auth_code );
```

The module figures out the right magic URL necessary for giving consent and obtaining the authorization token for us. Because I’m lazy, I pulled in a module that attempts to open a browser for you (works great on OS X, but your mileage may vary) with that URL. As I mentioned before, the end result is we are sitting at a Web page that says, “Please copy this code, switch to your application and paste it there: ...”. The next lines have our script prompting for that code and exchanging it for an access token.

Now let’s do the actual work with the Google Calendar API. The documentation at [5] is decent, but Google provides something even better, an awesome Web-based API explorer that shows all of the possible required and optional parameters and helps you try out various combinations before you actually code. This API explorer is linked off their Get Started page [6].

To actually get a list of calendars, we would do something like this:

```
my $response =
$oauth2->get("$cal_uri/users/me/calendarList");
die 'Could not retrieve cal list' unless
                         $response->is_success;

my $callist = decode_json $response->decoded_content;

foreach $cal ( @{ $callist->{items} } ) {
    print $cal->{summary} . ":\n" . $cal->{id} . "\n";
}
```

We make the proper GET request, and if it succeeds, we are handed back some JSON. We parse that JSON into Perl data structures and print a selected set of fields back from that info.

If that all makes sense, let’s take a look at the code for performing our real task. Our goal is to find all of the events that have some sort of notification set on them. Google’s calendar lets you set a default for the entire calendar for notifications; let’s make the assumption that the default for the calendar is sane (otherwise, we’d get alerted for a very large quantity of events because people don’t often change the default when they create an event). So now we have to locate the individual events that have a non-default notification set for them.

First, we’ll need to make sure we are looking at the right calendar. To read the events from a calendar, you have to request them from the calendar by referencing that calendar’s unique ID. That’s the reason why the previous sample prints out both a calendar’s summary (i.e., name) and its ID. The ID gets passed along in the request URL (along with some other parameters, more on that in a moment), so we’ll need to make sure it is URL-safe.

Here’s what it looks like to set an ID and make the original request for data (note, all of the OAuth2-related code is exactly the same as in the previous sample, so I’m only going to show the Calendar API-related code here):

```
my $calid = uri_escape 'somecalendarid@gmail.com';

$response =
$oauth2->get("$cal_uri/calendars/$calid/events?
              maxResults=100");

die 'Could not retrieve list of entries'
   unless $response->is_success;
```

You can see that the URL has changed and that we’ve added on a parameter to the URL itself. If you find you are using a number of parameters in the query, I recommend constructing the URL using the query_form() function from the URI module instead of doing it by hand as above.

So now we can print the results. Here we look for a special field in an event entry that indicates it is using a custom notification. If that exists, we print out the name of the event plus either the exact day and time it starts or just the day (if it is an all-day event).

```
foreach my $entry ( @{ $entries->{items} } ) {
      print "$entry->{summary} "
       . ( $entry->{start}->{dateTime} ||
           $entry->{start}->{date} ) . "\n"
      if exists $entry->{reminders}->{overrides};
}
```

So, we're done, right? Sorry, not so fast. If you have lots of entries in your calendar, you will have to deal with pagination. As we saw in the last column, the results come back *N* results at a time. By default, that number is 250, although you can raise it to 2500 according to the doc. I prefer to walk through the data in reasonable-sized pieces (100 at a time, that's what the maxResults parameter is doing there). With each result set (except for the last), Google hands back a nextPageToken that you can send in a subsequent request's pageToken parameter (note the different name!). Here's code that repeats the previous step for every subsequent page of data if there is any:

```
while ( $entries->{nextPageToken} ) {
    $response =
      $oauth2->get( "$cal_uri/calendars/$calid/events?"
          . "maxResults=100&"
          . "pageToken="
          . $entries->{nextPageToken} );
    die 'Could not retrieve addtl list of entries'
       unless $response->is_success;

    $entries = decode_json $response->decoded_content;

    foreach my $entry ( @{ $entries->{items} } ) {
        print "$entry->{summary} "
          . ( $entry->{start}->{dateTime} ||
                $entry->{start}->{date} ) . "\n"
          if exists $entry->{reminders}->{overrides};
    }
}
```

The key thing is that you send the exact same query again that yielded the paged result, the only difference being the addition of the pageToken parameter.

Now we are done. I hope this has given you a brief peek into how OAuth2 can be used to gain access to a real-live service. Take care, and I'll see you next time.

Resources

[1] Eran Hammer explains OAuth: http://hueniverse.com/2007/09/05/explaining-oauth/.

[2] Eran Hammer disclaiming OAuth2: http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/.

[3] Eran Hammer, NSFW, politics of OAuth2 process: https://vimeo.com/52882780.

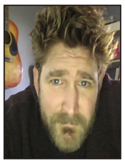
[4] Google docs on using OAuth2: https://developers.google.com/identity/protocols/OAuth2.

[5] Google docs for using the Calendar API: https://developers.google.com/google-apps/calendar/.

[6] Calendar API explorer: https://developers.google.com/google-apps/calendar/get_started.

iVoyeur
Tests and Metrics

DAVE JOSEPHSEN



Dave Josephsen is the sometime book-authoring developer evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

I realized this morning that I haven't been to a change management meeting in years. I imagine that many people who make software for a living haven't been to one in a decade or more. Continuous integration (CI) and automated testing killed change management for the modern software engineering shop, but those of us who fix things were left only with configuration management, which was not quite enough to keep us out of change meetings.

As I write this in the brief lull between AutomaCon, which finished last week, and LISA15, which will arrive before I'm ready for it, I can't help but muse a little bit on "infrastructure as code" (IaC)—the somewhat ungainly offspring of configuration management and continuous integration that has killed change management for me and increasing numbers of other operations folks.

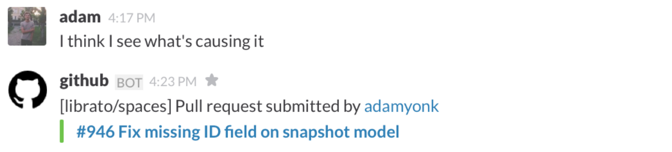
Setting aside for a moment the somewhat utopian notion of abstracting away all of our ugly pipes and wires into software, IaC is a good thing because it gives most of IT a common interface to make changes—namely, the deployment pipeline. Software engineers make changes to files that represent applications, and commit them to Git, which calls out to a CI tool to run some tests on it, and if they pass, a process or person deploys it to production. Now operations folks do pretty much the same thing, making changes to files that represent servers or routers or whatever, and committing them to Git and etc.

All of this rests on the foundation of continuous deployment, and continuous deployment rests on the foundation of tests. But testing infrastructure as code is a pretty new endeavor; it's just not something your typical sysadmin or operations person has much experience with. We're starting to see a few tools pop up, most notably Serverspec [1], but we're still going to need to become skilled in choosing and crafting good tests.

One thing I've noticed in my ongoing developer anthropology is that a lot of software engineers had and continue to have the same problem with choosing monitoring metrics. It's just not something the typical software engineer has a lot of experience with (which is tragic, but that's beside the point). And from that observation follows another: it turns out that choosing good tests and choosing good metrics are similar endeavors, and in this article, I'd like to explore some of those parallels with you.

Our Deployment Pipeline

Librato is a prolific engineering shop. We range between 40 and 60 deployments per day. In fact, as I write this, so far today we've deployed code 40 times—12 of which were production changes (the others targeted for various staging environments). I can see all of these deployments in our corporate chatroom, because we use chatbots to push code into production. In fact, most of our interaction with the services we maintain is abstracted behind chatbots in one way or another. So when someone merges some code into a production repo, I can see it in group-chat:


adam 4:17 PM
I think I see what's causing it

github BOT 4:23 PM
[librato/spaces] Pull request submitted by adamyonk
#946 Fix missing ID field on snapshot model

Figure 1: Chat transcript of a GitHub PR

And not only can I see the pull request (PR), I can see whether the proposed change passes or fails its unit tests:

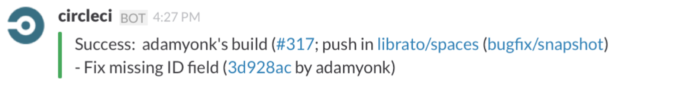
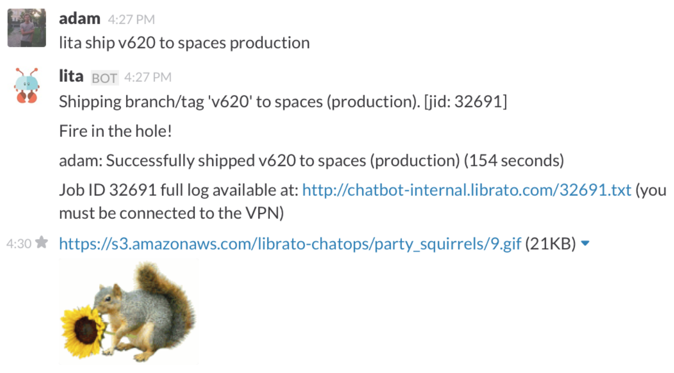

circleci BOT 4:27 PM
Success: adamyonk's build (#317; push in librato/spaces (bugfix/snapshot)
- Fix missing ID field (3d928ac by adamyonk)

Figure 2: Chat transcript of a passed unit test

And then I can watch with bated breath as the developer then deploys the change into production.


adam 4:27 PM
lita ship v620 to spaces production

lita BOT 4:27 PM
Shipping branch/tag 'v620' to spaces (production). [jid: 32691]
Fire in the hole!
adam: Successfully shipped v620 to spaces (production) (154 seconds)
Job ID 32691 full log available at: http://chatbot-internal.librato.com/32691.txt (you must be connected to the VPN)
4:30 https://s3.amazonaws.com/librato-chatops/party_squirrels/9.gif (21KB)

Figure 3: Chat transcript of a deployment

How Often Do Good Testers Test?

I'll pick on our alerting service because it's written in Go and, since it uses Go's built-in testing framework [2], is easy for a knuckle-dragger like myself to inspect with grep. Let's see how many test files there are:

```
find . | grep _test.go | wc -l
```

That returns 44 individual, test-laden files. Roughly one for every other .go-suffixed file in this repository, each one named for the unit it tests. Ergo, for foo.go, we find foo_test.go about half the time. Lightly poking into the files that don't have an associated test file, I find mostly type definitions and other data-structure-related code (not the sort of thing you normally test directly).

How about actual test functions?

```
grep -ri 'func Test.*(*testing.T' . | wc -l
```

This yields 172 individual tests. About a 4-1 ratio of total functions to test functions. So about 25% of the functions we create are tests.

What about by sheer volume of code?

```
find . | grep _test.go | while read i; do egrep -v '({|})' ${i} |
grep '[a-z][A-Z]'; done  | wc -l
```

Gives me close to 2400 lines of code devoted to tests. In fact, test-related code makes up almost half of this repository measured by lines. So OK, we test a lot, but then all of us who work in continuous integration shops do nowadays.

Change-control meetings are intended to protect healthy production environments from human error by instituting a layer of peer review. Whether this works or not is debatable, but it is unquestionably slow and drains productivity. Long release cycles allow more time for development and production branches to diverge. The classical change-control methodology, therefore, by slowing down the release cycle, tends to foster larger, more substantial (and therefore more error-prone) changes.

Relying on unit tests to protect us from human error instead allows us to make smaller, simpler, safer changes more often. We can spend as much time creating tests as we might otherwise spend on halting productivity to create change proposals and argue about them in a weekly meeting.

What Makes Good Tests Good?

The operative word there is RELY. Our tests can't protect the production environment if they aren't meaningful. In creating them, we generally need to be both procedural and selective. We need to select test criteria that we can genuinely rely on to help us ship quickly and safely.

GOOD TESTS ADD CONTEXT AND ENCOURAGE COOPERATION

If we make our tests too difficult, obtrusive, or meaningless, or if we try to enforce things like coding style that everyone hasn't already agreed to, people will just work around them. Self-defeating behavior like this is more likely to emerge when we sequester test creation to a particular team. Tests should mostly enforce the expected operational parameters of the things we create. Everyone should craft them, because they help us all reason about what we expect from the things we build. Tests that we didn't write should give us insight into new code-bases rather than encourage adversarial relationships between engineers.

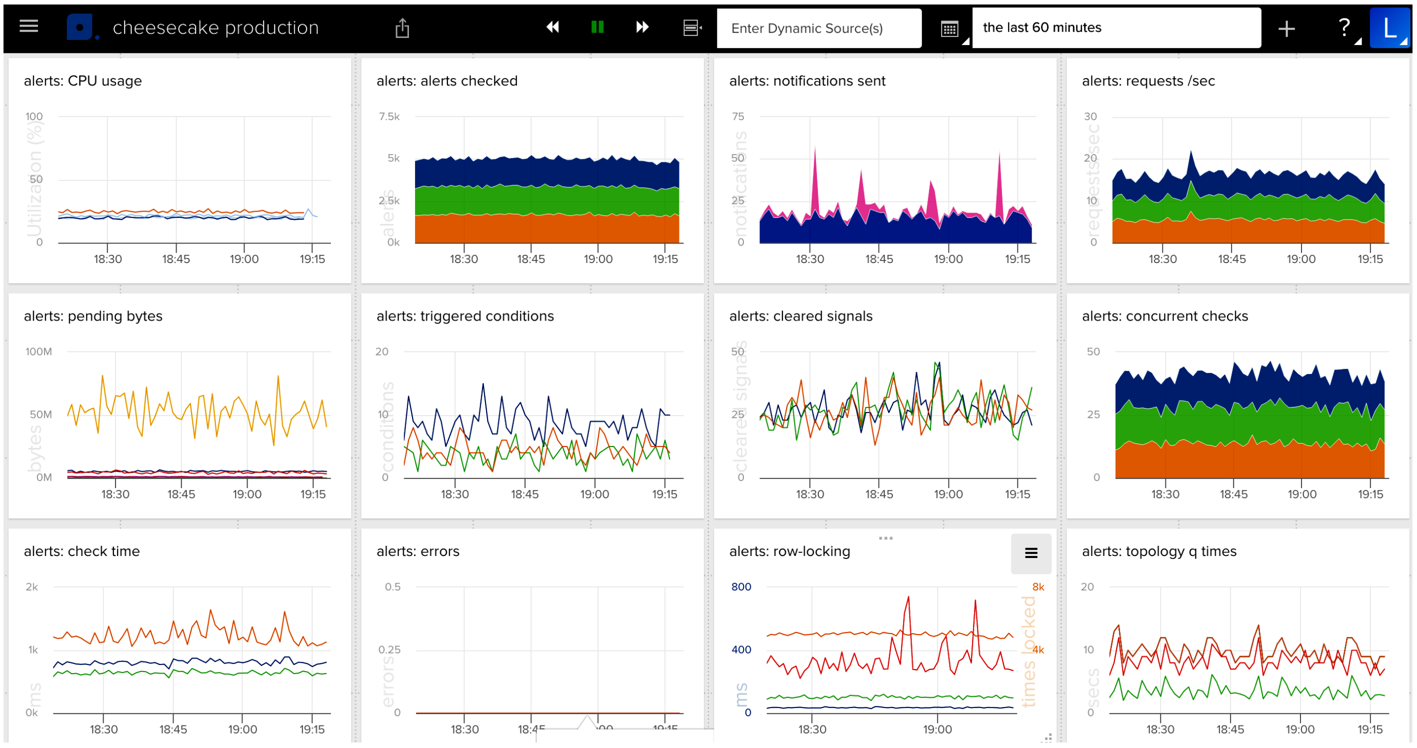


Figure 4: The alerting service dashboard

GOOD TESTS IMPROVE OUR DESIGN

Creating and maintaining good tests requires that we reason about correctness when we design and create software, thereby making us cognizant of our own expectations and assumptions. Choosing good test parameters means thoroughly understanding not only what we've created, but also the difference between what we've created and what we set out to create in the first place. Testable code is well-designed code, and poorly designed code is usually hard to test.

Metrics Are Tests that Never Stop Running

There's another class of code in this alerting repository that's neither functional to the application nor related to unit tests. An example looks something like this:

```
metrics.Measure("outlet.poll.alerts.count", len(alerts))
```

This is instrumentation code, and grep counts a little over 200 lines of it in this repository. The idea behind instrumentation is to measure important aspects of the application from within. Instrumentation like this quantifies things like queue sizes, worker-thread counts, inter-service latency, and request types. These metrics are then exported to a centralized system that helps us visualize the inner workings of our applications. In fact, here's a screenshot of the dashboard (Figure 4) where the metrics from this alerting service wind up.

Unit testing is like the sign at the theme park that says we need to be this tall before we can deploy to production. Our metrics are more like the canary in the coal mine. They are tests that can follow our code into production. They help us continuously vet our assumptions about the changes we introduce. Like test-driven development, which uses carefully crafted unit tests to verify correctness, metrics-driven development uses well-chosen metrics to directly show us the effect of our changes.

What Makes Good Metrics Good?

At this point I could en masse copy/paste the section I just wrote about what makes good tests good, substituting the word metric for test. Our metrics are the primary means by which we understand the behavior of our applications in the wild, and so we need to rely on them arguably even more than on our tests.

Like our tests, our metrics also help us protect the production environment from human error. If they aren't meaningful, our continuous integration pipeline suffers.

GOOD TESTS ADD CONTEXT AND ENCOURAGE COOPERATION

Good metrics test systems hypotheses. They confirm our expectations about how the things we build perform in real life. Just like tests, everyone should be able to choose and work with their own metrics because they help us all reason about what we

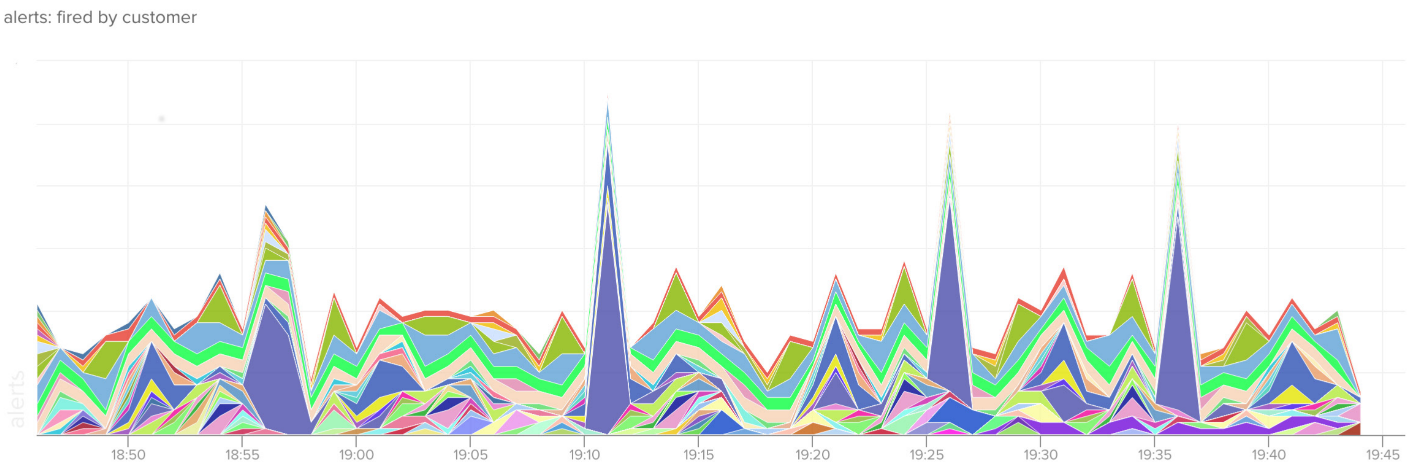


Figure 5: A well-chosen metric

expect from the things we build. Metrics can teach us about code bases that we aren't familiar with. Without any documentation whatsoever, I can infer many things from the metric in Figure 5 (e.g., this service sends alerts, the number of customers using it, the total and individual rates at which alerts are fired etc.).

GOOD METRICS MAKE GOOD CODEBASES

Choosing meaningful metrics also requires us to reason about correctness when we design and create software, but when we succeed, we gain ongoing operational insight that's invaluable to everyone, whether they're designing systems, regression testing, supporting infrastructure, or shipping features.

Good instrumentation is a sign of operational health. It keeps us cognizant of our own expectations and assumptions. Well-measured code is usually well-designed code, and poorly designed code is usually difficult to measure.

So even if you've never created a unit test and find the notion of Serverspec and IaC daunting, you can take comfort in the reality that being a diligent student of systems monitoring and reading excellently crafted columns like this one has prepared you for what is to come. No need to thank me, I'm already positively drowning in acclaim.

Take it easy.

Resources

[1] Serverspec: http://serverspec.org/.

[2] Testing in Go: https://golang.org/pkg/testing.

For Good Measure
Why Speculate?

DAN GEER

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

The late Michael Crichton was many things. He had both extraordinary imagination and unquivering analytic clarity. In this column, I borrow the title and more from his magnificent essay, "Why Speculate?" given in La Jolla, California, at the International Leadership Forum on April 26, 2002 [1].

Boiled down, Crichton simply said that no one knows the future and that those who pretend to do so are self-serving, delusional, or something else equivalently uncomplimentary. So are the people who believe what the predictors say. He notes how big the prediction industry really is, singling out media especially, and he reminds us all that the track record for sweeping predictions is pretty poor. He coins a clinical term, and I might as well copy his text where he does so:

> Media carries with it a credibility that is totally undeserved. You have all experienced this in what I call the Murray Gell-Mann Amnesia effect. (I refer to it by this name because I once discussed it with Murray Gell-Mann, and by dropping a famous name I imply greater importance to myself, and to the effect, than it would otherwise have.)
>
> Briefly stated, the Gell-Mann Amnesia effect is as follows. You open the newspaper to an article on some subject you know well. In Murray's case, physics. In mine, show business. You read the article and see the journalist has absolutely no understanding of either the facts or the issues. Often, the article is so wrong it actually presents the story backward—reversing cause and effect. I call these the "wet streets cause rain" stories. Paper's full of them.
>
> In any case, you read with exasperation or amusement the multiple errors in a story, and then turn the page to national or international affairs, and read as if the rest of the newspaper was somehow more accurate about Palestine than the baloney you just read. You turn the page, and forget what you know.
>
> That is the Gell-Mann Amnesia effect. I'd point out it does not operate in other arenas of life. In ordinary life, if somebody consistently exaggerates or lies to you, you soon discount everything they say. In court, there is the legal doctrine of falsus in uno, falsus in omnibus, which means untruthful in one part, untruthful in all. But when it comes to the media, we believe against evidence that it is probably worth our time to read other parts of the paper. When, in fact, it almost certainly isn't. The only possible explanation for our behavior is amnesia.

Everyone reading this article knows precisely what Crichton is talking about (or was, 13 years ago): what is written about cybersecurity for the general audience is often counterfactual and/or counterlogical. Unfortunately, what is written for specific audiences like legislatures and regulatory agencies is also counterfactual and/or counterlogical. And all of this finds an audience because of an actual need that I argue is acutely important for cybersecurity—we need to predict the future if our tools are to intersect our problems on target and in time.

That is the theme here—that the fast-moving nature and, yes, the unpredictability of the cybersecurity regime are such that were it occasionally possible to make useful predictions, we would be better off, better able to accomplish our security plans while those plans were still relevant. At the same time, and especially in cybersecurity, no one can predict the future. We desperately need prediction, we know it, and it is impossible to do and increasingly so.

I am, myself, entirely guilty of trying to do prediction in cybersecurity. I give speeches on this precisely [2]. I am working on a personal project right now whose only point is prediction. With a quant colleague, I've long run another. I work on the periphery of the intelligence community, and the intelligence community is entirely about prediction—constantly speculating on what is our actual position and what is our actual velocity. If your very job is security in any sense, then you want all the prediction you can get.

Yet, at the same time, surprises happen. If he were still with us, Crichton would remind us that "[T]he problem with speculation is that it piggybacks on the Gell-Mann effect of unwarranted credibility, making the speculation look more useful than it is." One can argue that compliance is a predictive exercise, based on the idea that "if you do this thing, then you can approach the future with less to fear." I buy that train of thought wholeheartedly, but what if the rules to be complied with cannot keep up with the rate of change? If they cannot, then whatever the prediction of outcome that compliance promises is prediction made relative to conditions that no longer hold. That can't be good. Or useful.

Unpredictability is so true in cybersecurity that we have a special name for when prediction fails: zero-day. We accept that a genuine 0day is an attack that no one could have seen coming. We so very often imply that failing to handle that 0day is blameless since, after all, it was not predicted. Yet every time a particularly lurid 0day shows up, I find myself thinking, "Could I have predicted that? How?"

In my last column [3], I leaned on Nassim Taleb's writing to relate how "the fat tails of power law distributions enlarge the variance of our estimates leading to less frequent but more severe failures (*The Black Swan*). The best one could say is that most days will be better and better but some will be worse than ever. Everything with a power law underneath has that property, and cyberspace's interconnectivity and interdependence are inherently power law phenomena." A fat-tailed setting inherently resists prediction, but for that very reason makes prediction ever more attractive to pursue.

So we get published predictions. Lots of them. Many of them hedge their bets by phrasing their prediction as a question, but that only invokes Betteridge's Law of Headlines ("Any headline that ends in a question mark can be answered by the word no").

It's a quandary. Fast change means tool sets for protection always trail the need unless the need can be forecast. Fast change makes forecasts hard if that fast change is one of adding mechanisms, not just scale, to the equation. We've got both scale (IoT with a 35% compound annual growth rate) and mechanism (afterthought interconnection of sundry gizmos runs on the proliferation of mechanism).

To be deadly serious about cybersecurity requires that *either* we damp down the rate of change, slowing it enough to give prediction operational validity—OR—we purposely increase unpredictability so that opposition targeting grows too hard for them to do. In the former, we give up various sorts of progress. In the latter, we give up various sorts of freedom as it would be the machines then in charge, not us.

But look at that; I can't even talk about prediction without making a prediction...

References

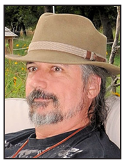
[1] Michael Crichton's 2002 speech: http://geer.tinho.net/crichton.why.speculate.txt.

[2] Dan Geer, "What Does the Future Hold for Cyber Security?" Suits and Spooks: http://geer.tinho.net/geer.suitsandspooks.19vi15.txt.

[3] Dan Geer, "The Denominator," *;login:*, vol. 40, no. 5, October 2015: https://www.usenix.org/publications/login/oct15/geer.

/dev/random
Streamailer and Beyond

ROBERT G. FERRELL



Robert G. Ferrell is an award-winning author of humor, fantasy, and science fiction, most recently *The Tol Chronicles* (www.thetolchronicles.com).

I once regarded Snapchat as not much more than a novelty: interesting and well-intentioned, perhaps, but a novelty nonetheless. It seemed only a matter of time before an app or service invalidated the ephemeral nature of its content and therefore the sum total of its raison d'être. At the same moment, though, I thought perhaps having email that disappeared after a few minutes might be nice, too.

That led my fiction-writer's brain to pondering how spies could send one another short-lived messages by writing them out and taking pictures of them to be sent via Snapchat, or for more obscurity, embedding the message in said photos steganographically, thus giving the receiving party only a short time to decode them. I'm relatively certain this has already been done somewhere.

Getting back to the disappearing email idea, I was idly speculating one day (because energetic speculation has taken its toll on me over the years) that while I've tended to archive everything, there's really no need for this now. The vast majority of non-spam emails these days are just alternatives to text messaging; that is, they only convey information that is relevant for a comparatively short time.

The behavior this ephemeral nature drives, of course, is to delete mails as soon as you've read them, but this still leaves behind some artifacts in the form of temp files and "deleted email" boxes that presuppose you might want to "undelete" some of these from time to time. We've all deleted something we wish we hadn't at one point or another. But what if you're the type who sends messages you really, really don't want anyone but the recipient to see, ever?

My answer is Streamailer. In this killer app, every incoming mail is assigned a probability of being read according to its origin address information and subject matter. Mail messages stream constantly; you choose which ones to pluck out of the digital river. Messages from people you have marked as friends stream by more frequently, and are assigned to a different buffer. Others circle by only a set number of times if you don't actively delete them. The buffers are bit-complement overwritten at the end of each spool cycle. Not totally unrecoverable, but most people who might be snooping your personal business don't have a scanning tunnel electron microscope in their garage. At least, not one that actually works.

Moving on, it is time once again to poke at the squirming, slime-covered electric eel that is infosec. When I look out over the vast hysterical cybersecurity wasteland these days with the jaundiced eye of one who has retired from the field after many years of banging my head against any nearby hard surface as a result of the recalcitrant idiots I encountered, I have to chuckle. (I think I should get some sort of award for the preceding sentence.) What was back when I started in the 1980s an obscure IT subspecialty dismissed by most corporate and governmental honchos with the wave of an illegal Cuban cigar has now become an 800-pound gorilla sitting on the developed world's chest, beating its own and grunting loudly.

There is a certain satisfaction to be gained from having been one of a handful of information security people who can say "we told you this stuff was important 25 years ago," but it's attenuated by the fact that the associated excreta have impacted the ventilation system blades in a very big way. Identity theft, encryption munging, mobster-in-the-middle attacks, ransomware, spear-phishing…that litany of horrors makes Aleph's "Smashing the Stack for Fun and Profit" seem positively wholesome by comparison.

In 1994 I was working as the one-man IT department for a small-to-medium research company in Austin, Texas. I had been registering domain names for early Web adopters (my NIC handle was RGF4, to give you some idea of the time frame), and I had just accomplished same for my employer. I developed their first (text-based; hooray for Lynx) Web page, then added graphics to it when Mosaic was released. I had to create said graphics using MS Paint to build bitmaps pixel by pixel.

At that time I was administering a couple of Novell networks (50-node 3.11 and 3.12, if I recall correctly) for our internal data sharing. I had been using email on UNIX systems for over ten years by then, and I was aware of what a powerful and soon-to-be-ubiquitous communications medium it was. Once I had finally won the battle to register a domain name for us and develop a rudimentary Web presence, I started in trying to convince management that SMTP was the next logical step. I developed a PowerPoint presentation (yes, we had PP even back then) that showed all the nifty things we could accomplish if only we had email. He clutched the coiled wire-sporting handset of his beloved rotary dial desk phone with white knuckles and refused to consider my heretical proposal. SMTP, in his mind, could never hope to compete with POTS.

I did finally drag him into the email age, but it was a lengthy and exhausting battle. This same person, I should add, for years resisted any effort to install a network drop in his office. I would wire one in during the evening and the next morning he would put tape over it and order it removed. He was still using MS DOS 5 for all work as late as 1995. He would print out memos and have his secretary photocopy them for distribution. I half expected the mail room boy to come by and ask if I'd heard the new Frank Sinatra single.

If there truly is one, the moral underlying this little story is that it often requires a great deal of time, effort, and frustration to convince those in charge that they need to take action or change direction. I could point to the climate change debacle currently underway as another prime example of leaders who don't want to face up to an inconvenient reality, but I won't. Instead, I will finish up by abruptly changing the subject.

I have a new proposal for personal authentication. Instead of tired old passwords, security questions regarding your childhood best friend's favorite dog, and shining lasers up your nostrils, I submit that we should instead be using what I will call vocabulary profiles. Present people with recordings of a series of, say, ten words chosen randomly from a list. Ask them to spell each. The unique way in which a person (mis)spells ten relatively difficult words would make a decent identifier.

But why choose this method over any number of similar and equivalent ones? Because watching people misspell "misspelled" and "illiterate" is one of the little joys in the otherwise colorless life of a struggling novelist.

Book Reviews

MARK LAMOURINE AND RIK FARROW

Python for Data Analysis
Wes McKinney
O'Reilly Media, 2013, 447 pages
ISBN 978-1-449-31979-3
Reviewed by Mark Lamourine

Python has become a popular programming language for numeric processing. This is largely due to the creation of NumPy, a binary module that provides the performance of a compiled program with the ease of use and fast development cycle of an interpreted language.

NumPy was just the start. A range of modules have been added on top to provide even more specialized functions or different techniques for accessing and manipulating numerical data. Wes McKinney is actively developing an extension called "pandas" (the name is not capitalized). According to the introduction on the pandas section of pydata.org (http://pandas.pydata.org), "pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with 'relational' or 'labeled' data both easy and intuitive." That's important because NumPy by itself wasn't designed to work with complex data structures composed of mixed data in the columns.

In *Python for Data Analysis*, McKinney offers a primer for data analysis with Python, using iPython for interaction, and NumPy and pandas for numerical manipulation and data access.

McKinney opens with a whirlwind tour of simple examples to showcase the capabilities of Python, iPython, NumPy and pandas, along with matplotlib and SciPy. He then returns to the beginning to add depth to each topic. His treatment of iPython for interactive development is one of the best I've seen. I learned several things I hadn't known before. The chapter on NumPy basics is similarly informative.

When he gets to using pandas I must say I began to lose the thread. The examples are clear and effective in showing the creation of and operations on the data structures, but I don't have the background in data analysis to follow why they are important. I can only assume that someone with more experience would know when and why to use them. Operations for loading and storing data are clear to me, but data transformations, aggregation, and group operations are outside my scope. The section on generating graphics with matplotlib was also in my range, but when McKinney uses it to display financial and time-series data, I appreciate the images but not necessarily their significance.

In the final section, McKinney returns to advanced features of NumPy, including some internals and optimizations.

Python for Data Analysis is not going to be a good place for someone new to numerical analysis. For readers already familiar with the concepts but who need to learn to do their work using Python, it's going to be a great resource.

An Introduction to Programming in Go
Caleb Doxsey
CreateSpace Independent Publishing Platform, 2012, 161 pages
ISBN 978-1478355823
Reviewed by Mark Lamourine

Google (in the persons of Robert Griesemer, Rob Pike, Ken Thompson) began development of Go in 2007, announced it publicly in 2009, and released version 1.0 in 2012. Go is distributed under the BSD license. At about the same time, according to the copyright notice, Caleb Doxsey was releasing *An Introduction to Programming in Go* on the Web (www.golang-book.com/books/intro), in PDF, and in hard copy. The copyright page notes that portions are used under the Creative Commons 3.0 Attribution license, originally published by Google, but I can't find any indications of which parts.

In this slim volume Doxsey provides a brief look into all of the major features of the Go language and of the development environment: installation, compilation, syntax and language features, a model for testing, software packages, and a survey of the core standard libraries. Each chapter closes with a set of exercises that should be easy for an experienced developer, and an appropriate challenge for a classroom student.

The hard copy clearly shows its heritage as a Web document. The typography and layout almost feel like a large-text book, perhaps typeset originally with LaTeX. The book was published using the CreateSpace Independent Publishing Platform, a service offered by Amazon. I've reviewed other books created using alternative publishers, and I'm encouraged that I can buy books like this that might never have made it to print under a traditional imprint. Amazon might not be to some people's liking but it is one choice for an unsigned author.

The independent publishing origin doesn't detract at all from the book's utility as a primer for new Go programmers. This is yet another case where I appreciate a paper book even when the complete text is available online. The Web version shows a copyright date of 2015, so it's likely that Doxsey is continuing to make updates, though a quick scan didn't show anything obvious. Go recently released version 1.5 and that includes several backward-compatible syntax changes that could make the examples simpler or clearer.

An Introduction to Programming in Go is by no means a complete, in-depth tour of the Go language, but as a first look it is a good choice.

The Go Programming Language
Alan Donovan and Brian Kernighan
Addison-Wesley Professional Computing Series, 2015, 400 pages
ISBN 978-0134190440
Reviewed by Rik Farrow

I still have Kernighan and Ritchie's *The C Programming Language* sitting on the bookshelf within reach of my desk. What made that book so useful was its clarity, brevity, and a wealth of examples that were actually relevant to a lot of the tasks one might have to handle in C.

Alan Donovan and Brian Kernighan have written an even better book. Like the original "white book," they include relevant examples all through the book's 400 pages. What's much better is that we now have the Internet, and all of the examples can be built using `go build`, or downloaded for editing, and that's very useful when it comes time to try out the exercises.

I don't think anyone can learn a programming language without actually working with it, and the path Donovan and Kernighan provide is a reasonable one. The first chapter, "Tutorial," covers examples for things you can learn in the next seven chapters. That means you can't possibly understand all that's covered in Chapter one, but you get an overview of what will be described eventually in greater depth.

And there is a lot of depth. I had no idea how different Go was, although I had some inklings. One of the things I liked about the book is that the authors casually repeat key points—for example, that variables, functions, and types that you want to export between packages must begin with a capital letter. This, and other subtleties, can easily trip up someone new to a programming language, and I found the reminders helpful.

One-fifth of the book is devoted to just two topics: interfaces and goroutines. Interfaces are contracts, promises in the strongly typed Go, that provide much of the flexibility that has been exposed in the earlier six chapters. And goroutines, along with channels for communicating between goroutines, provide the primary, but not the only, method for writing concurrent applications. The ninth chapter covers more traditional techniques for concurrency using shared variables.

I enjoyed reading this book, which is not something I can say about most technical books. The style is always direct and understandable, yet at the same time, I felt like I was reading for an advanced class in programming, what I might have encountered in my third year of university. You can learn about more than just Go by reading this book. If you have any interest in learning Go, I strongly recommend that you own this book.

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

Free subscription to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, networks, and operating systems, book reviews, and reports of sessions at USENIX conferences.

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

Access to videos from USENIX events in the first six months after the event: www.usenix.org/publications/multimedia/

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.

2016 Election for the USENIX Board of Directors

by Casey Henderson, USENIX Executive Director

The biennial election for officers and directors of the Association will be held in the spring of 2016. A report from the Nominating Committee is now available on the USENIX Web site at www.usenix.org/board/elections16. USENIX members will receive notification of this report via email.

Nominations from the membership are open until January 4, 2016. To nominate an individual, send a written statement of nomination signed by at least five (5) members in good standing, or five separately signed nominations for the same person, to the Executive Director at the Association offices, to be received by noon PST, January 4, 2016. Please prepare a plain-text Candidate's Statement and send both the statement and a photograph (minimum size 1800 pixels by 1200 pixels) to production@usenix.org, to be included in the ballots.

Ballots will be mailed to all paid-up members in early February 2016. Ballots must be received in the USENIX offices by March 21, 2016. The results of the election will be announced on the USENIX Web site by March 30 and will be published in the Summer 2016 issue of *;login:*.

The Board consists of eight directors, four of whom are "at large." The others are the president, vice president, secretary, and treasurer. The balloting is preferential: those candidates with the largest numbers of votes are elected. Ties in elections for directors shall result in run-off elections, the results of which shall be determined by a majority of the votes cast. Newly elected directors will take office at the conclusion of the first regularly scheduled meeting following the election, or on July 1, 2016, whichever comes earlier.

Team USA Continues to Impress at IOI 2015 in Kazakhstan

Brian C. Dean, Director of the USA Computing Olympiad

I am always amazed at the computing talent shown by the top high school students in this country.

This summer, I traveled to Almaty, Kazakhstan, to lead a team of high-school computer-science students—the top four from the entire USA—to compete at the 27th International Olympiad in Informatics (IOI), the most prestigious computing contest in the world at the high-school level. Aside from myself and deputy leaders Mark Gordon and Amy Quispe, team USA this year consisted of:

- Daniel Chiu, a sophomore from Catlin Gable High School in Oregon
- Demi Guo, a junior studying abroad at Hangzhou No. 2 High School in China
- Andrew He, a senior from Monta Vista High School in California
- Alexander Wei, a junior from Phillips Exeter Academy in New Hampshire

Joining us in this week-long event were 318 other students representing 82 countries, all eager to put their algorithmic programming skills to the test to vie for glory in the form of gold, silver, and bronze medals. I am thrilled to report that our team had one of its best showings ever: three gold medals (Chiu, He, Wei) and one silver (Guo). Gold medals are only awarded to the top 1/12 of all participants, and no other country earned more than three, putting team USA right at the top alongside other powerhouse countries such as China, South Korea, and Russia. Andrew He even earned third place individually, improving on his gold medal performance at IOI 2014.

The competition format at the IOI involves two five-hour contests, each asking students to code solutions for three challenging problems of an algorithmic nature. The difficulty is exceedingly high—at a level that would probably stump most graduate students in computing. Only one competitor earned a perfect score this year, Jeehak Yoon from South Korea. As an example of one of the problems, suppose you need to deliver boxes from a depot to N different houses, all situated at different locations on a long circular road. You have at your disposal a truck that can hold only K boxes at a time, and you want to determine the minimum driving distance needed to deliver every box. The values of N and K can be in the millions, and your program needs to run in less than a second to receive full marks. Every student on team USA received a perfect score on this problem, one of the easier problems in the contest.

Aside from the competition, our IOI hosts organized a variety of cultural activities, excursions, social events, and other activities that helped expose everyone to Kazakh culture and tradition. My favorite excursion was into the nearby mountains, where a half-hour cable car ride brought us to a vantage point 10,000 feet high with stunning views of the city below. We also had a chance to experience Kazakh cuisine, where I discovered new foods that tasted fairly good (e.g., horse meat), and others that I'm fairly certain may be more of an acquired taste (e.g., camel milk). During the opening and closing ceremonies we were treated to a wide range of Kazakh song and dance. The entire week was a wonderful experience for all involved, and we look forward to IOI 2016, to be held in Kazan, Russia.

The reason team USA has performed so well at recent IOIs is largely due to the rigorous selection and training process we use to create our team each year, overseen by a national organization called the USA Computing Olympiad (USACO). The USACO is a non-profit organization that provides online training material and programming contests for students of all ages interested in learning algorithmic problem solving. Thousands take part in our contests each year, starting with our easiest "bronze" division problems that require basic programming ability but no specific algorithmic knowledge. Successful students are promoted to the "silver" division, where contest problems help them learn standard algorithmic techniques. Those who excel in silver are finally promoted to our "gold" division, featuring our most challenging problems (roughly on par with IOI problems). Each year we invite the top two dozen gold competitors in the USA to attend a rigorous summer training camp at Clemson University, from which the team of four is ultimately selected to attend the IOI. Training camp is a whirlwind experience packed with practice contests, advanced lectures, and fun side activities. Our goal with camp (and also with the USACO in general) is not only to train a winning team to attend the IOI, but to inspire students about computing as a discipline and to encourage them to fill the ranks of the next generation of top computer scientists.

The USACO is run by a small but dedicated group of volunteer coaches and supported entirely by funds provided by our corporate sponsors. I am exceedingly grateful for our support from USENIX, one of our strongest and most loyal sponsors. This support has enabled our program to grow and evolve, nearly doubling in size in the past five years alone. Based on the impressive accomplishments of our alums to date, your sponsorship has created a measureable and substantial impact in cutting-edge computing, both in academia and industry.

This year was particularly poignant for our organization, due to the passing of Dr. Donald Piele, who founded the USACO nearly two decades ago. Don posthumously received the IOI's "distinguished service award" to recognize his contributions to the IOI community. Don's early work with the USACO has inspired countless students—myself included—to pursue careers in computing, and I am honored to be able to continue leading the organization so that it may continue to inspire others.

To the USENIX community: thank you again for your outstanding support for high-school computing, and I look forward to reporting continued good results from IOI 2016 and beyond! To learn more about the USACO, please visit our Web site at usaco.org.

Thanks to Our Volunteers

by Casey Henderson, USENIX Executive Director

As many of our members know, USENIX's success is attributable to a large number of volunteers who lend their expertise and support for our conferences, publications, good works, and member services. They work closely with our staff in bringing you the best in the fields of systems research and system administration. Many of you have participated on program committees, steering committees, and subcommittees, as well as contributing to this magazine. The entire USENIX staff and I are most grateful to you all. Below, I would like to make special mention of some people who made particularly significant contributions in 2015.

Program Chairs

13th USENIX Conference on File and Storage Technologies (FAST '15)
Jiri Schindler and Erez Zadok

2015 USENIX Research in Linux File and Storage Technologies Summit (Linux FAST Summit '15)
Ric Wheeler

12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)
Paul Barham and Arvind Krishnamurthy

SREcon15
Sabrina Farmer, Andrew Fong, and Fernanda Weiden

SREcon15 Europe
Narayan Desai and John Looney

15th Workshop on Hot Topics in Operating Systems (HotOS XV)
George Candea

2015 USENIX Annual Technical Conference (ATC '15)
Shan Lu and Erik Riedel

7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '15)
Irfan Ahmad and Tim Kraska

7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '15)
Ken Salem and John Strunk

24th USENIX Security Symposium (USENIX Security '15)
Jaeyeon Jung

9th USENIX Workshop on Offensive Technologies (WOOT '15)
Aurélien Francillon and Thomas Ptacek

2015 USENIX Journal of Election Technology and Systems Workshop (JETS '15)
Walter Mebane and Dan S. Wallach, Program Chairs and Editors-in-Chief, *USENIX Journal of Election Technology and Systems (JETS)*

8th Workshop on Cyber Security Experimentation and Test (CSET '15)
Adam Aviv and Iulian Neamtiu

5th USENIX Workshop on Free and Open Communications on the Internet (FOCI '15)
Masashi Crete-Nishihata and Phillipa Gill

2015 USENIX Workshop on Health Information Technologies (HealthTech '15)
Apu Kapadia and David Kotz

2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE '15)
Mark Gondree and Zachary N J Peterson

2015 USENIX Summit on Hot Topics in Security (HotSec '15)
Joseph Bonneau and Carrie Gates

29th Large Installation System Administration Conference (LISA15)
Cory Lueninghoener and Amy Rich

2015 USENIX Release Engineering Summit (URES '15)
Gareth Bowles and Dinah McNutt

2015 USENIX Container Management Summit (UCMS '15)
Matthew Barr and Ashley Penney

Other Chairs and Major Contributors

FAST '15
Poster Chair: Donald Porter
Tutorial Coordinator: John Strunk

Linux FAST Summit '15
Organizational Assistance: Christoph Hellwig and Theodore Ts'o

NSDI '15
Poster Session Co-Chairs: Rama Ramasubramanian and Franziska Roesner

USENIX Security '15
Deputy Program Chair: Thorsten Holz
Invited Talks Committee: Michael Bailey, Angelos Keromytis (Chair), Damon McCoy, and Gary McGraw
Poster Session Co-Chairs: Adam Doupé and Sarah Meiklejohn
Work-in-Progress Reports (WiPs) Coordinator: Tadayoshi Kohno

LISA15
Invited Talks Co-Chairs: Doug Hughes and Mario Obejas
Academic Co-Chairs: Paul Anderson and Marc Chiarini
Workshops Chair and Lightning Talks Coordinator: Lee Damon
LISA Lab Co-Chairs: Tony Del Porto and Andrew Mundy
LISA Build Coordinators: Branson Matheson and Brett Thorson

Storage Pavilion and Data Storage Day at LISA15
Organizer: Jacob Farmer of Cambridge Computer

2015 USENIX Journal of Education in System Administration (JESA)
Editors-in-Chief: Kyrre Begnum and Charles Border

USENIX Board of Directors
Cat Allman, John Arrasjid, David Blank-Edelman, Daniel V. Klein, Brian Noble, Kurt Opsahl, Carolyn Rowland, and Hakim Weatherspoon

Audit Committee
Cat Allman, John Arrasjid, and Niels Provos

Awards Committee
Brian Noble and Matt Simmons

Development Advisory Committee
Cat Allman, John Arrasjid, Brian Noble, Kurt Opsahl, and Hakim Weatherspoon

USA Computing Olympiad (co-sponsored by USENIX)
Team Leader: Brian Dean
Deputy Team Leaders: Mark Gordon and Amy Quispe

HotCRP Submissions and Reviewing System
Eddie Kohler

USENIX and LISA Blogger
Ben Cotton

;login: Goes Quarterly

by Casey Henderson, USENIX Executive Director

In 2016, *;login:* is taking the next step in its long history: It will change from a bimonthly to a quarterly schedule, with four issues per year.

Prior to becoming a bimonthly magazine in 1997, *;login:* was a bimonthly newsletter, albeit a beefy one. At its inception, *;login:* truly was a newsletter, tracing its roots to the "USENIX NEWS" pamphlets we've been revisiting to celebrate USENIX's 40th anniversary. This December 2015 issue of *;login:* is nearly unrecognizable when compared to those early missives. It's largely to the credit of Rik Farrow, our editor for many years, that the magazine you're currently reading is such a thorough celebration of the state of the art across the many communities that USENIX represents.

For each issue, Rik gathers content by attending conferences, contacting authors of papers or articles touching on the latest research and practice in our field, and revisiting connections he's gathered and maintained during his years as a key member of the community. Then our production team, led by managing editor Michele Nelson, takes over to lay out and produce this volume. The *;login:* production cycle never ends, because we're working on the next issue well before the current one arrives in your mailbox. A quarterly schedule will stretch out these periods to be much saner for our busy staff, while still delivering a similar amount of high quality content you receive annually. You can expect each quarterly issue to be lengthier than the bimonthly issues you've been receiving.

For those of you who are USENIX members, you're aware that *;login:* is one of our primary membership benefits. While *;login:* officially accounts for $90 of each member's annual dues, rising production costs are straining the budget. We have not raised dues for many years—since 2008, in fact—in order to keep membership as affordable as possible. Though we had considered an increase in membership dues for 2016, I have recommended to the USENIX Board that we continue to maintain the current rate, partially in light of this change in *;login:'s* number of issues.

Although there are many people involved in *;login:'s* success, from the authors of articles to the typesetter, please join me in thanking Rik and Michele in particular for their dedication to keeping *;login:* the top-notch publication it is. They're both excited about the extra breathing room this schedule will offer them, including the opportunity to not work on *;login:* while ostensibly on vacation. If *;login:* is your vacation reading material of choice, you can enjoy the magazine even more knowing that its creators are now able to have a similar opportunity to relax.

CoolDC '16: USENIX Workshop on Cool Topics in Sustainable Data Centers

March 19, 2016 • Santa Clara, CA

Sponsored by USENIX, the Advanced Computing Systems Association

The USENIX Workshop on Cool Topics in Sustainable Data Centers (CoolDC '16) will take place on March 19, 2016, directly following NSDI '16 in Santa Clara, CA.

Important Dates

- Paper submissions due: Tuesday, December 15, 2015, 8:59 p.m. PST
- Notification to authors: Tuesday, February 2, 2016
- Final paper files due: Tuesday, March 1, 2016

Workshop Organizers

Program Co-Chairs

Weisong Shi, *Wayne State University*
Thomas F. Wenisch, *University of Michigan*

Program Committee

Kirk Cameron, *Virginia Tech*
Christina Delimitrou, *Stanford University*
Michael Ferdman, *Stony Brook University*
David Irwin, *University of Massachusetts Amherst*
Tao Li, *University of Florida/NSF*
Jie Liu, *Microsoft Research*
Chris Malone, *Google*
Karthick Rajamani, *IBM Research*
Anand Sivasubramaniam, *The Pennsylvania State University*
Xiaorui Wang, *The Ohio State University*
Qiang Wu, *Facebook*
Zhe Zhang, *Cloudera*

Overview

Around the mid-2000s, the advent of mega-scale internet services and public cloud offerings led to a redesign of data center architectures which addressed key inefficiencies, particularly in electrical and mechanical infrastructure. At the same time, accelerated need for efficient servers spurred a generation of research on CPU, memory, network, and storage power management techniques, which have led to a marked improvement in server efficiency and energy proportionality. However, this first generation of improvement has plateaued; further opportunity in the large-scale mechanical infrastructure is limited, and no single server or network component stands out as the key source of inefficiency. Hence, it is time for a second, holistic, clean-slate redesign of the data center, encompassing new server architectures, heterogeneous computing platforms, radical networking paradigms, new mechanical and electrical designs, intelligent cluster management, and radical rethinking of software architectures while considering changing usage patterns (e.g., hybrid private/public clouds).

In addition to developing promising technologies to improve data center efficiency, we also need new metrics to assess the success of SDC research. Currently, power usage effectiveness (PUE) is a widely reported metric to assess the energy efficiency of a data center. The impact of renewables can be assessed via carbon usage effectiveness (CUE) to measure the combined impact of clean energy and energy efficiency on greenhouse gas emissions, and water usage effectiveness (WUE) can be used to assess the water usage of a data center. And yet, all three of these metrics fall short of describing the true efficiency of the data center. They fail to reflect waste at the enclosure/tray level (e.g., VRMs, server fans). Moreover, they do not assess the efficiency or value of the computation being performed and hence fail to reflect server hardware inefficiencies or software bloat.

The 2016 USENIX Workshop on Cool Topics in Sustainable Data Centers (CoolDC '16) is a forum to disseminate results and stimulate further cutting-edge research in quantitative design, evaluation, and research methods for sustainable data centers. The goal of the workshop is to become a venue where experts in sustainable energy systems, data center physical infrastructure, networking and server architecture, cloud computing, and internet-scale applications can come together to exchange ideas on how to maintain and improve the sustainability of warehouse-scale computer infrastructure.

Topics

Topics of interest in sustainable data centers include but are not limited to:

- Instrumentation, measurement, and characterization studies
- Metrics, benchmarks, interfaces
- Performance, energy and other resource trade-offs, energy complexity
- Energy-efficient software optimization, application design
- System-level optimization, cross-layer coordination
- Scheduling, run-time adaptation, feedback control

- Processor, memory, network, storage, hardware components and architecture
- Reliability and power management
- Thermal management
- Green energy sources and their implications
- Technologies for and management of energy storage
- Life-cycle analysis

The workshop seeks submissions of early-stage research and novel ideas that have a high likelihood of generating interesting discussion.

Submission Instructions

Please submit your papers by 8:59 p.m. PST on December 15, 2015. Papers must be in PDF format and must be submitted via the Web submission form linked from the Call for Papers Web site, www.usenix.org/cooldc16/cfp. Do not email submissions.

Submitted papers must be no longer than 6 single-spaced 8.5" x 11" pages. The complete submission should be typeset in two-column format in 10-point type on 12-point (single-spaced) leading, with the text block being no more than 6.5" wide by 9" deep. Submissions that violate any of these restrictions may not be reviewed. The limits will be interpreted fairly strictly, and no extensions will be given for reformatting. If you wish, you may use our LaTeX templates and style files, available at www.usenix.org/conferences/author-resources/paper-templates.

Reviewing will be double-blind; therefore, please do not include any author names on any submitted documents except in the space provided on the submission form. You must also ensure that the metadata included in the PDF does not give away the authors. If you are improving upon your prior work, refer to your prior work in the third person and include a full citation for the work in the bibliography. For example, if you are building on your own prior work in the papers [1, 2, 3], you would say something like: "While prior work did X, Y, and Z [1, 2, 3], this paper additionally does W, and is therefore much better." Do NOT omit or anonymize references for blind review.

Submissions to CoolDC '16 may not be under consideration for any other venue. 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/submissions-policy for details.

Questions? Contact your program co-chairs, cooldc16chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Papers accompanied by nondisclosure agreement forms will not be considered. Accepted submissions will be treated as confidential prior to publication on the CoolDC '16 Web site; rejected submissions will be permanently treated as confidential.

All papers will be available online to registered attendees before the workshop. 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 day of the workshop, March 19, 2016.

2016 USENIX Annual Technical Conference

June 22–24, 2016, Denver, CO

Sponsored by USENIX, the Advanced Computing Systems Association

Important Dates

- Paper submissions due: **Monday, February 1, 2016, 11:59 p.m. GMT**
- Notification to authors: **Friday, April 15, 2016**
- Final paper files due: **Tuesday, May 24, 2016**

Conference Organizers

Overview

Authors are invited to submit original and innovative papers to the Refereed Papers Track of the 2016 USENIX Annual Technical Conference. We seek high-quality submissions that further the knowledge and understanding of modern computing systems with an emphasis on implementations and experimental results. We encourage papers that break new ground, present insightful results based on practical experience with computer systems, or are important, independent reproductions/refutations of the experimental results of prior work. USENIX ATC '16 has a broad scope, and specific topics of interest include (but are not limited to):

- Architectural interaction
- Big data infrastructure
- Cloud computing
- Datacenter networking
- Deployment experience
- Distributed and parallel systems
- Embedded systems
- Energy/power management
- File and storage systems
- Mobile and wireless
- Networking and network services
- Operating systems
- Reliability, availability, and scalability
- Security, privacy, and trust
- System and network management and troubleshooting
- Usage studies and workload characterization
- Virtualization

USENIX ATC '16 is especially interested in papers broadly focusing on practical techniques for building better software systems: ideas or approaches that provide practical solutions to significant issues facing practitioners. This includes all aspects of system development: techniques for developing systems software; analyzing programs and finding bugs; making systems more efficient, secure, and reliable; and deploying systems and auditing their security.

Experience reports and operations-oriented studies, as well as other work that studies software artifacts, introduces new data sets of practical interest, or impacts the implementation of software components in areas of active interest to the community are well-suited for the conference.

The conference seeks both long-format papers consisting of 11 pages and short-format papers of 5 pages, not including references. Short papers will be included in the proceedings, and will be presented as normal but in sessions with slightly shorter time limits.

For industrial practitioners, if you are interested in the Practitioner Talks Track, which accepts proposals for 20-minute or 40-minute talks, please refer to the USENIX ATC '16 Call for Talks Web page at www.usenix.org/atc16/cft.

Best Paper Awards

Cash prizes will be awarded to the best papers at the conference. Please see www.usenix.org/conferences/best-papers for Best Paper winners from previous years.

Best of the Rest Track

The USENIX Annual Technical Conference is the senior USENIX forum covering the full range of technical research in systems software. Over the past two decades, USENIX has added a range of more specialized conferences. ATC is proud of the content being published by its sibling USENIX conferences and will be bringing a track of encore presentations to ATC '16. This "Best of the Rest" track will allow attendees to sample the full range of systems software research in one forum, offering both novel ATC presentations and encore presentations from recent offerings of ATC's sibling conferences.

What to Submit

Authors are required to submit full papers by the paper submission deadline. *It is a hard deadline; no extensions will be given.* All submissions for USENIX ATC '16 will be electronic, in PDF format, via the Web submission form on the Call for Papers Web site, www.usenix.org/atc16/cfp.

USENIX ATC '16 will accept two types of papers:

Full papers: Submitted papers must be no longer than 11 single-spaced 8.5" x 11" pages, including figures and tables, but **not** including references. You may include any number of pages for references. Papers should be formatted in 2 columns, using 10-point type on 12-point leading, in a 6.5" x 9" text block. Figures and tables must be large enough to be legible when printed on 8.5" x 11" paper. Color may be used, but the paper should remain readable when printed in monochrome. The first page of the paper should include the paper title and author name(s); reviewing is single blind. Papers longer than 11 pages, **not including references,** or violating formatting specifications will not be reviewed. In a good paper, the authors will have:

- Addressed a significant problem
- Devised an interesting and practical solution or provided an important, independent, and experimental reproduction/refutation of prior solutions
- Clearly described what they have and have not implemented
- Demonstrated the benefits of their solution
- Articulated the advances beyond previous work
- Drawn appropriate conclusions

Short papers: Authors with a contribution for which a full paper is not appropriate may submit short papers of at most 5 pages, **not including references,** with the same formatting guidelines as full papers. You may include any number of pages for references. Examples of short paper contributions include:

- Original or unconventional ideas at a preliminary stage of development
- The presentation of interesting results that do not require a full-length paper, such as negative results or experimental validation
- Advocacy of a controversial position or fresh approach

For more details on the submission process and for templates to use with LaTeX and Word, authors should consult the detailed submission requirements linked from the Call for Papers Web site. Specific questions about submissions may be sent to atc16chairs@usenix.org.

By default, all papers will be made 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. In any case, the papers will be available online to everyone beginning on the first day of the conference, June 22, 2016.

Papers accompanied by nondisclosure agreement forms will not be considered. Accepted submissions will be treated as confidential prior to publication on the USENIX ATC '16 Web site; rejected submissions will be permanently treated as confidential.

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/submissions-policy for details.

Note that the above does not preclude the submission of a regular full paper that overlaps with a previous short paper or workshop paper. However, any submission that derives from an earlier paper must provide a significant new contribution (for example, by providing a more complete evaluation), and must explicitly mention the contributions of the submission over the earlier paper. If you have questions, contact your program co-chairs, atc16chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Authors will be notified of paper acceptance or rejection by April 15, 2016. Acceptance will typically be conditional, subject to shepherding by a program committee member.

Poster Session

The poster session is an excellent forum to discuss ideas and get useful feedback from the community. Posters and demos for the poster session will be selected from all the full paper and short paper submissions by the poster session chair. If you do not want your submissions to be considered for the poster session, please specify on the submission Web site.

Program and Registration Information

Complete program and registration information will be available in April 2016 on the conference Web site.

Rev. 10/29/15

2016 USENIX Annual Technical Conference

June 22–24, 2016, Denver, CO

Sponsored by USENIX, the Advanced Computing Systems Association

Important Dates

- Talk submissions due: Monday, February 1, 2016, 11:59 p.m. GMT
- Notification to submitters: Friday, April 15, 2016

Conference Organizers

Overview

Industrial practitioners are invited to submit talk proposals to the Practitioner Talks Track of the 2016 USENIX Annual Technical Conference. The USENIX Annual Technical Conference is the senior USENIX forum covering the full range of technical research in systems software. This track seeks presentations about practical solutions and challenges to significant real-world issues facing industrial practitioners. It will provide a unique venue for industrial and academia participants to exchange ideas and experiences.

Examples of talk topics include, but are not limited to:

- Techniques that solve significant issues in practice
- Tool development and problem-solving experience report
- Forgotten research topics that are highly relevant to industry
- New challenges faced by industrial practitioners that need help from research
- Interesting data set or benchmark suite available for the community

Examples of technical areas include, but are not limited to:

- Architectural interaction
- Big data infrastructure
- Cloud computing
- Datacenter networking
- Deployment experience
- Distributed and parallel systems
- Embedded systems
- Energy/power management
- File and storage systems
- Mobile and wireless
- Networking and network services
- Operating systems
- Reliability, availability, and scalability
- Security, privacy, and trust
- System and network management and troubleshooting
- Usage studies and workload characterization
- Virtualization

The content of the talk will not be included in the conference proceedings. If you are interested in the Refereed Papers Track, which accepts submissions of 11 pages, not including references, and 5 pages, not including references, please refer to the USENIX ATC '16 Call for Papers Web page at www.usenix.org/atc16/cfp.

What to Submit

Talk proposals must include the following and be submitted before the submission deadline to receive full consideration.

- Title: Should make it obvious what your talk is about
- Description: Include attendee takeaways and why people want to hear this talk; if applicable, please also provide white papers or Web pages or videos that support this talk
- Should: Please indicate the talk topic and area of interests
- Speaker: Include past public speaking experience, with a URL to past presentations if available
- Duration: Talk or tutorial length (either 20 or 40 minutes)

All submissions will be through the Web submission form on the Call for Talks Web site, www.usenix.org/atc16/cft.

Presenting a talk that you gave before in another venue is allowed. However, any talk proposal that derives from an earlier talk must explicitly mention where and when the earlier talk was given in the talk description. If you have questions, contact your program co-chairs, atc16chairs@usenix.org.

Submitters will be notified of talk acceptance or rejection by April 15, 2016.

Poster Session

The poster session is an excellent forum to discuss ideas and get useful feedback from the community. If you plan to prepare a poster for your talk, and want your poster to be considered for the poster session, please specify so in the submission form.

Program and Registration Information

Complete program and registration information will be available in April 2016 on the conference Web site.

Questions?

Contact atc16chairs@usenix.org.

Rev. 10/29/15

Statement of Ownership, Management, and Circulation, 10/1/15

Title: *;login:* Pub. No. 0008-334. Frequency: Bimonthly. Number of issues published annually: 6. Subscription price $90.

Office of publication: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

Headquarters of General Business Office of Publisher: Same. Publisher: Same.

Editor: Rik Farrow; Managing Editor: Michele Nelson, located at office of publication.

Owner: USENIX Association. Mailing address: As above.

Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, or other securities: None.

The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes have not changed during the preceding 12 months.

Extent and Nature of Circulation			Average No. Copies Each Issue During Preceding 12 Months	No. Copies of Single Issue (August 2015) Published Nearest to Filing Date
a. Total Number of Copies			2938	2700
b. Paid Circulation	(1)	Outside-County Mail Subscriptions	1392	1366
	(2)	In-County Subscriptions	0	0
	(3)	Other Non-USPS Paid Distribution	739	773
	(4)	Other Classes	0	0
c. Total Paid Circulation			2131	2139
d. Free Distribution By Mail	(1)	Outside-County	0	0
	(2)	In-County	0	0
	(3)	Other Classes Mailed Through the USPS	80	60
	(4)	Free Distribution Outside the Mail	425	300
e. Total Free Distribution			505	360
f. Total Distribution			2636	2499
g. Copies not distributed			302	201
h. Total			2938	2700
i. Percent Paid			81%	86%
Paid Electronic Copies			378	339
Total Paid Print Copies			2509	2478
Total Print Distribution			3014	2838
Percent Paid (Both Print and Electronic Copies)			83%	88%

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

usenix 40TH ANNIVERSARY

ENIGMA

It's time for the security community to take a step back and get a fresh perspective on threat assessment and attacks. This is why the USENIX Association is excited to announce the launch of Enigma, a new security conference geared towards those working in both industry and research.

Expect three full days of high-quality speakers, content, and engagement for which USENIX events are known.

The full program and registration are now available.

enigma.usenix.org

JANUARY 25–27, 2016

SAN FRANCISCO, CALIFORNIA, USA

usenix ASSOCIATION