;login:

WINTER 2017 VOL. 42, NO. 4

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

Enigma 2018
January 16–18, 2018, Santa Clara, CA, USA
www.usenix.org/enigma2018

FAST '18: 16th USENIX Conference on File and Storage Technologies
February 12–15, 2018, Oakland, CA, USA
www.usenix.org/fast18

SREcon18 Americas
March 27–29, 2018, Santa Clara, CA, USA
www.usenix.org/srecon18americas

NSDI '18: 15th USENIX Symposium on Networked Systems Design and Implementation
April 9–11, 2018, Renton, WA, USA
www.usenix.org/nsdi18

SREcon18 Asia/Australia
June 6–8, 2018, Singapore
www.usenix.org/srecon18asia

USENIX ATC '18: 2018 USENIX Annual Technical Conference
July 11–13, 2018, Boston, MA, USA
Submissions due February 6, 2018
www.usenix.org/atc18

Co-located with USENIX ATC '18

HotStorage '18: 10th USENIX Workshop on Hot Topics in Storage and File Systems
July 9–10, 2018
Submissions due March 15, 2018
www.usenix.org/hotstorage18

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

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

USENIX Security '18: 27th USENIX Security Symposium
August 15–17, 2018, Baltimore, MD, USA
Submissions due February 8, 2018
www.usenix.org/sec18

Co-located with USENIX Security '18

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

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

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

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

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

HotSec '18: 2018 USENIX Summit on Hot Topics in Security
August 14, 2018
www.usenix.org/hotsec18

SREcon18 Europe/Middle East/Africa
August 29–31, 2018, Dusseldorf, Germany
www.usenix.org/srecon18europe

OSDI '18: 13th USENIX Symposium on Operating Systems Design and Implementation
October 8–10, 2018, Carlsbad, CA, USA
Abstract registration due April 26, 2018
www.usenix.org/osdi18

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

;login:

WINTER 2017 VOL. 42, NO. 4

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Michele Nelson
michele@usenix.org

COPY EDITORS
Steve Gilmartin
Amber Ankerholz

PRODUCTION
Arnold Gatilao
Jasmine Murcia

TYPESETTER
Star Type
startype@comcast.net

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

www.usenix.org

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

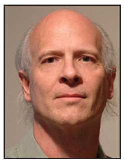
$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for nonmembers are $90 per year. Periodicals postage paid at Berkeley, CA, and additional mailing offices.

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

EDITORIAL

Musings

RIK FARROW



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

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

Hardware

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

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

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

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

There was an alternative design, a mainframe built in the mid-to-late '60s: the GE 645 (later Honeywell). The GE 645 had segment registers, essentially hardware used to add an offset to each memory address. Unlike memory management, segment registers as used in this design weren't limited to large page sizes, so it was possible to have programs that could treat different areas of memory as if they were different physical compartments, and limit access to those compartments. Please read my interview with Peter G. Neumann in this issue for more on segmentation.

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

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

Software

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

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

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

People

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

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

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

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

The Lineup

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

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

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

The last article in this series, where the ability of people to behave securely is in question, is about checking password strength. Melicher et al. developed a neural network small enough to download as part of a Web page, and proved their tool (https://github.com/cupslab/password_meter) works much better than the current crop of tools, which reward people for capitalizing the first letter and ending their password with "!".

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

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

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

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

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

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

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

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

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

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

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

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

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

Conclusion

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

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

References

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

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

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

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

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

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

SAVE THE DATES!

SRECON® AMERICAS

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

SRECON® ASIA AUSTRALIA

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

SRECON® EUROPE MIDDLE EAST AFRICA

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

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

Follow us at @srecon

usenix®
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SECURITY

Global-Scale Measurement of DNS Manipulation

PAUL PEARCE, BEN JONES, FRANK LI, ROYA ENSAFI, NICK FEAMSTER, NICHOLAS WEAVER, AND VERN PAXSON

Paul Pearce is a senior PhD student at UC Berkeley advised by Vern Paxson and is a member of the Center for Evidence-based Security Research (CESR). His research brings empirical grounding to Internet security problems, including censorship, cyber crime, and advanced persistent threats (APTs). pearce@cs.berkeley.edu

Ben Jones is a Software Engineer at Google and a PhD candidate at Princeton University. His research is in the area of Internet censorship measurement. He holds a BS in computer science from Clemson University and was an Open Technology Fund Senior Fellow in Information Controls. bj6@cs.princeton.edu

Frank Li is a PhD student at the University of California, Berkeley. His research mainly focuses on improving the remediation process for security issues such as vulnerabilities and misconfigurations. More broadly, he is interested in large-scale network measurements and empirical studies in a computer security context. frankli@cs.berkeley.edu

Roya Ensafi is a Research Assistant Professor in Computer Science and Engineering at the University of Michigan, where her research focuses on computer networking and security. She pioneered the use of side-channels to remotely measure network interference and censorship of Internet traffic. Prior to joining Michigan, she was a postdoc at Princeton University. ensafi@umich.edu

Despite the pervasive and continually evolving nature of Internet censorship, measurements remain comparatively sparse. Understanding the scope, scale, and evolution of Internet censorship requires global measurements, performed at regular intervals. We developed Iris, a scalable, accurate, and ethical method to continually measure global manipulation of DNS resolutions. Iris reveals widespread DNS manipulation of many domain names across numerous countries worldwide.

Anecdotes and reports indicate that Internet censorship is widespread, affecting at least 60 countries [5]. Despite pervasive Internet censorship, empirical Internet measurements revealing the scope of that censorship remain sparse. A more complete understanding of Internet censorship around the world requires *diverse* measurements from a wide range of geographic regions and ISPs, not only across countries but also within regions of a single country.

Unfortunately, most mechanisms for measuring Internet censorship rely on volunteers who run measurement software deployed on their own Internet-connected devices (e.g., laptops, phones, tablets) [9, 10]. Because these tools rely on individuals performing actions, their scale and diversity are fundamentally limited.

Although recent work has developed techniques to continuously measure widespread manipulation at the transport [4, 7] and HTTP [1] layers, a significant gap remains in understanding global information control via manipulation of the Internet's Domain Name System (DNS). Towards this goal, we developed and deploy Iris [8], a method and system to ethically detect, measure, and characterize the manipulation of DNS responses within countries across the entire world—without involving users or volunteers.

Iris aims to provide sound assessments of potential DNS manipulation indicative of an intent to restrict user access to content. To achieve high detection accuracy, we rely on a set of metrics that we base on the underlying properties of DNS domains, resolutions, and infrastructure. Using our implementation of Iris, we performed a global measurement study that highlights the heterogeneity of DNS manipulation across resolvers, domains, and countries—and even within a country.

One significant design challenge concerns ethics. In contrast to systems that explicitly involve volunteers in collecting measurements, methods that perform censorship measurement without volunteers raise the issue of user risk. To this end, Iris ensures that, to the extent possible, we only involve Internet *infrastructure* (e.g., within Internet service providers or cloud hosting providers) in an attempt to minimize the risk to individual users.

How and What to Measure?

Detecting DNS manipulation is conceptually simple: perform DNS queries through geographically distributed DNS resolvers and analyze the results for "incorrect" responses that indicate manipulation of the answers. Despite this apparent simplicity, realizing a system to scalably collect DNS data and analyze it for manipulation poses significant technical and ethical challenges. Technically, how do we acquire or find global vantage points? Once we

Nick Feamster is a Professor in the Computer Science Department at Princeton University and the Deputy Director of the Princeton University Center for Information Technology Policy (CITP). He received his PhD in computer science from MIT in 2005 and his SB and MEng degrees in electrical engineering and computer science from MIT in 2000 and 2001, respectively. His research focuses on many aspects of computer networking and networked systems, with a focus on network operations, network security, and censorship-resistant communication systems. feamster@cs.princeton.edu

Nicholas Weaver is a Computer Security Researcher at the International Computer Science Institute and a Lecturer in the Computer Science Department at UC Berkeley. nweaver@icsi.berkeley.edu

Vern Paxson is a Professor of Electrical Engineering and Computer Sciences at UC Berkeley and leads the Networking and Security Group at the International Computer Science Institute in Berkeley. His research focuses heavily on measurement-based analysis of network activity and Internet attacks. He works extensively on high performance network monitoring, detection algorithms, cybercrime, and countering censorship and abusive surveillance. vern@berkeley.edu

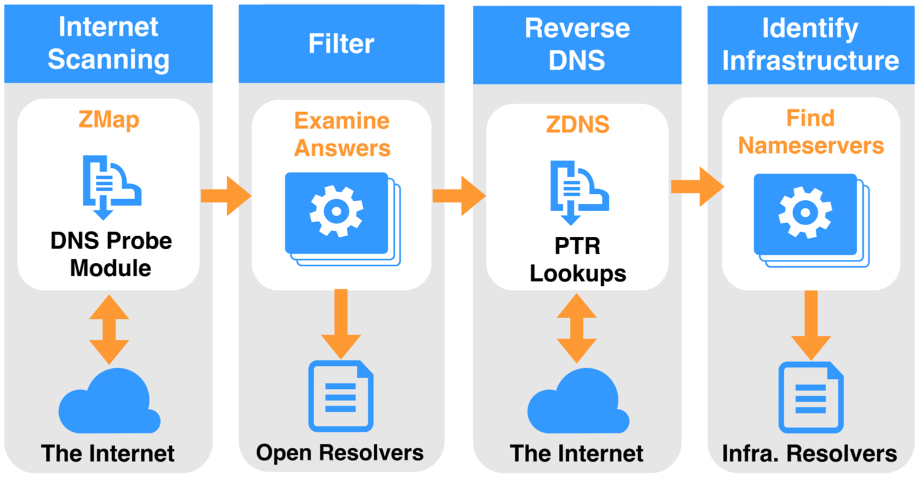


Figure 1: Overview of Iris's DNS resolver identification and selection pipeline

have them, what DNS names should we measure? Ethically, how do we conduct wide-ranging third-party measurements without implicating innocent citizens? What steps should we take to ensure that Iris does not induce undue load on the DNS resolution infrastructure?

Finding Vantage Points

To obtain a wide range of measurement vantage points, we leverage *open DNS resolvers* deployed around the world; such resolvers will resolve queries for any client.

Measurement using open DNS resolvers entails complex ethical considerations. Given their prevalence and global diversity, open resolvers offer a compelling resource, providing researchers with extensive volume and reach. Unfortunately, open resolvers also pose risks not only to the Internet but to individual users. For example, resolvers may operate in an open fashion as a result of configuration errors; they frequently operate on end-user devices such as home routers [6]. Using these devices for measurement can impose monetary costs and, if the measurement involves sensitive content or hosts, can also expose their owners to harm.

Due to these ethical considerations, we restrict the set of open resolvers that we use to a small subset of resolvers for which we have high confidence they are part of the Internet infrastructure, as opposed to attributable to particular individuals. Figure 1 illustrates the process by which Iris finds safe open DNS resolvers.

Conceptually, the process of finding infrastructure resolvers has two steps: (1) scanning the Internet for open DNS resolvers and (2) pruning the list of open DNS resolvers that we identify to limit the resolvers to a set that we can plausibly attribute to Internet infrastructure.

Step 1: Scanning the Internet's IPv4 space for open DNS resolvers. Scanning the IPv4 address space provides a global perspective on open resolvers. To do so, we developed an open-source module for the ZMap network scanner [3] to enable Internet-wide DNS resolutions. This module queries port 53 of all IPv4 addresses with a recursive DNS A record query. We use a purpose-registered domain name we control for these queries to ensure there is a known correct answer. From these scans, we conclude that all IP addresses that return the correct answer to this query are open resolvers.

Step 2: Identifying infrastructure resolvers. We prune the set of all open DNS resolvers on the Internet to include only those resolvers that appear to function as authoritative nameservers for some DNS domain. Iris uses the ZDNS tool to perform reverse DNS PTR lookups

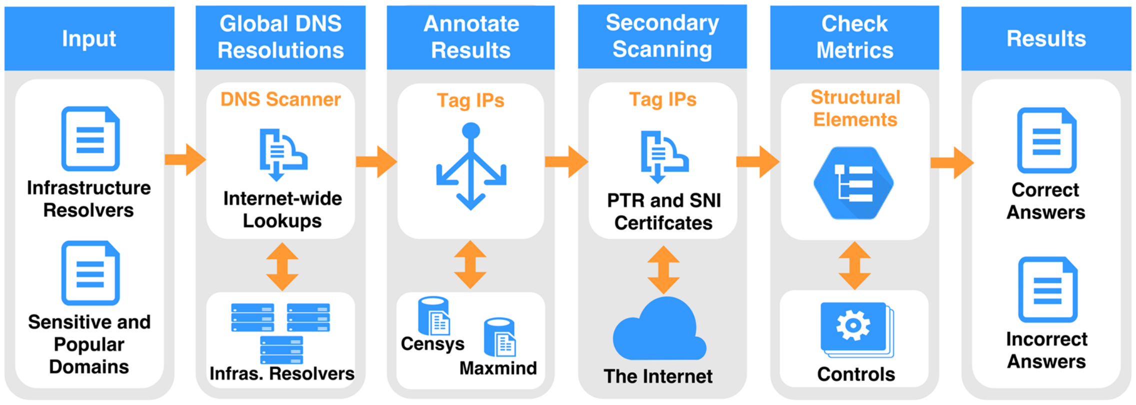


Figure 2: Overview of DNS resolution, annotation, filtering, and classification. Iris takes as input a set of domains and DNS resolvers and outputs results indicating manipulated DNS responses.

for all open resolvers and retains only the resolvers that have a valid PTR record beginning with the subdomain `ns[0-9]+` or `nameserver[0-9]*`. This filtering step reduces the number of usable open resolvers from millions to thousands; fortunately, even the remaining set of resolvers suffices to provide broad country- and network-level coverage.

Because we conduct our measurements using resolvers we do not control, we cannot differentiate between countrywide or state-mandated censorship and localized manipulation at individual resolvers (e.g., captive portals or malware [6]). To mitigate this uncertainty, we aggregate our measurements to ISP or country granularity.

Ethical Reasoning

Our primary ethical concern is minimizing the risks associated with issuing DNS queries via open resolvers for potentially censored or manipulated domains, as these DNS queries could implicate innocent users. A second concern is the query load that Iris induces on authoritative nameservers. With these concerns in mind, we use the ethical guidelines of the Belmont Report and Menlo Report to frame our discussion. One important ethical principle is *respect for persons*; essentially, an experiment should respect the rights of humans as autonomous decision-makers. In lieu of attempting to obtain informed consent, we draw upon the principle of *beneficence*, which weighs the benefits of conducting an experiment against the risks associated with the experiment. We note that the benefit of issuing DNS queries through tens of millions of resolvers has rapidly diminishing returns, and that using only open resolvers that we can determine are unlikely to correspond to individual users greatly reduces the risk to any individual without dramatically reducing the benefits of our experiment.

An additional guideline concerns *respect for law and public interest*, which essentially extends the principle of beneficence to all relevant stakeholders, not only the experiment participants. To abide by this principle, we rate-limit our queries for each domain to ensure that the owners of the domains do not face significant expenses as a result of the queries that we issue.

Which DNS Domains to Query

Iris queries a list of sensitive URLs published by Citizen Lab, known as the Citizen Lab Block List (CLBL). This list of URLs is compiled by experts based on known censorship around the world, labeled by category. We distill the URLs down to domain names and use this list as the basis of our data set. We then supplement the list by adding additional domain names selected at random from the Alexa Top 10,000. These additional domain names help address geographic or content biases in the CLBL while maintaining a manageable total number of queries.

Iris: Systematic Detection of DNS Manipulation

We describe Iris's process for issuing queries for the domains to the set of usable open resolvers. Figure 2 provides an overview of the process. Iris resolves each DNS domain using the global vantage points afforded by the open DNS resolvers; annotates the responses with information from both auxiliary data sets as well as additional active probing; and uses consistency and independent verifiability metrics to identify manipulated responses. A more in-depth treatment of this topic appeared at USENIX Security 2017 [8].

Collecting Annotated DNS Responses

Performing Global DNS Resolutions

Iris takes as input a list of suitable open DNS resolvers as well as the combined CLBL and Alexa domain names. We also include three DNS domains under our control to allow us to compute consistency metrics. Querying tens of thousands of domains across tens of thousands of resolvers required the development of a new DNS query tool, because no existing DNS measurement tool supported this scale. We implemented this aspect of Iris in 649 lines of Go. The tool takes as input a set of domains and resolvers and coordinates randomized querying of each domain across each resolver. To prevent overloading resolvers and domains, we randomize domain order and maintain strict upper bounds on how fast Iris queries individual resolvers.

Annotating DNS Responses with Auxiliary Information
Our analysis ultimately relies on characterizing both the *consistency* and the *independent verifiability* of the DNS responses that we receive. To enable this classification, we first must gather additional details about the IP addresses returned in each of the DNS responses. Iris annotates each IP address returned in the set of DNS responses with additional information about geolocation, autonomous system (AS), port 80 HTTP responses, and port 443 HTTPS X.509 certificates. We rely on Censys [2] for daily snapshots of this auxiliary information, and further annotate IP addresses with AS and geolocation information from the MaxMind service, as applicable.

Additional PTR and TLS Scanning
For each IP address, we perform a DNS PTR lookup to facilitate additional consistency checks. Complicating inference further, a single IP address might host multiple Web sites via HTTP or HTTPS (virtual hosting). To mitigate this effect, we perform an active HTTPS connection to each returned IP address using the Server Name Indication (SNI) to specify the name originally queried.

Identifying DNS Manipulation
To determine whether a DNS response is manipulated, Iris relies on two metrics: *consistency* and *independent verifiability*.

Unmanipulated access to a domain should manifest some degree of *consistency*, even when accessed from varying global vantage points. The consistency may take the form of network properties, infrastructure attributes, or content. We leverage these attributes, both in relation to control data as well as across the data set itself, to classify DNS responses.

Our consistency metric relies on access to a set of geographically diverse resolvers that we control and are presumably not subject to manipulation. These control resolvers return a set of answers that we can presume are correct and thus can use to identify consistency across a range of IP address properties. Geographic diversity helps ensure that area-specific deployments do not cause false positives. We also use control domains to test whether a resolver reliably returns unmanipulated results for non-sensitive content (e.g., not a captive portal).

For each domain name measured, we create a set of consistency metrics by taking the union of each metric across all of our control resolvers. For example, we consider an answer *consistent* if the IP address matches at least one seen by any of our controls.

In addition to consistency metrics, we also define a set of metrics based on HTTPS certificate infrastructure that we can independently verify using external data sources. This data is collected both from both auxiliary annotations and active HTTPS SNI scans.

We say that a response is correct if it satisfies *any* consistency or independent verifiability metric; otherwise, we classify the response as *manipulated*.

Global Measurement Study
Using Iris, we performed an end-to-end global measurement study of DNS manipulation during January 2017. Here we describe the basic composition and statistics of this measurement study.

Resolvers
We initially identified a large pool of open DNS resolvers through an Internet-wide ZMap scan using a custom DNS measurement extension to ZMap that we developed. In total, 4.2 million open resolvers responded with a correct answer to our scan queries. This number excludes 670K resolvers that replied with correctly formed DNS responses but with either a missing or an incorrect answer for the scan's query domain.

The degree to which we can investigate DNS manipulation across various countries depends on the geographic distribution of the selected DNS resolvers. By geolocating this initial set of resolvers, we observed that the pool spanned 232 countries and dependent territories, with a median of 659 resolvers per country. Abiding by our ethical considerations reduced this set to 6,564 infrastructure resolvers in 157 countries, with a median of six resolvers per country. Finally, we removed unstable or otherwise errant resolvers, further reducing the set of usable resolvers to 6,020 in 151 countries, again with a median of six resolvers in each. While our final set of resolvers is a small fraction of all open DNS resolvers, it still suffices to provide a global perspective on DNS manipulation.

Domains
We began with the CLBL, consisting of 1,376 sensitive domains. We augmented this list with 1,000 domains randomly selected from the Alexa Top 10,000, as well as the three control domains that we manage that we do not expect to be manipulated. Due to overlap between the two domain sets, our combined data set consisted of 2,330 domains. We excluded 27 problematic domains (e.g., domains that had expired or had broken authoritative name servers) that we identified through our data collection process, resulting in a final set of 2,303 domains.

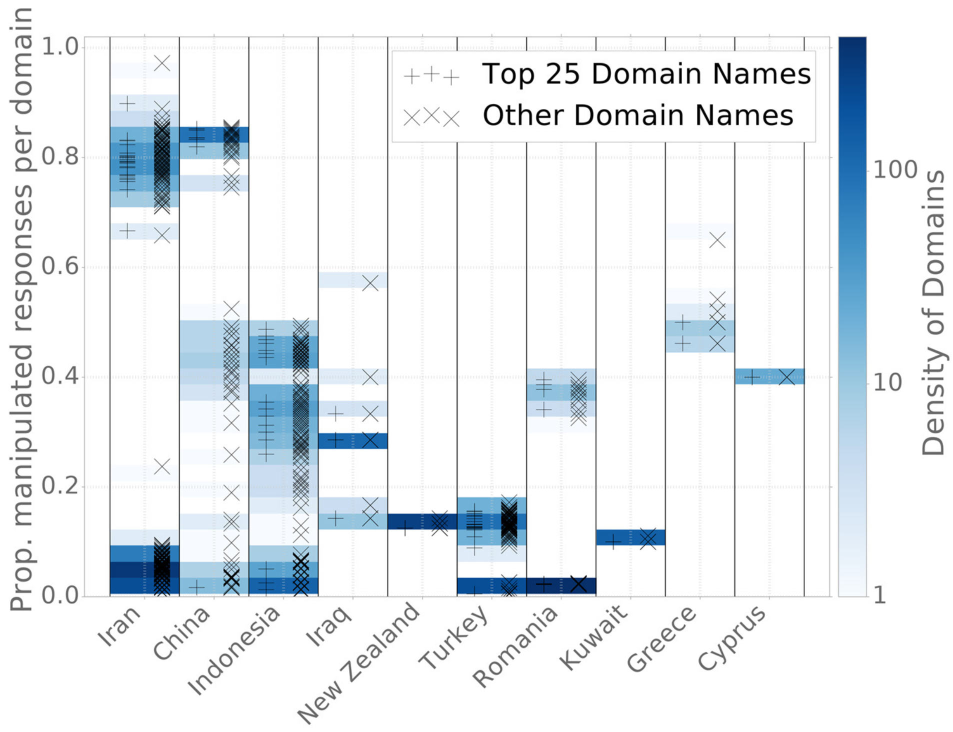


Figure 3: The fraction of resolvers within a country that manipulate each domain

Total Queries

We issued 14.5 million DNS A record queries during a two-day period in January 2017. After removing problematic resolvers, domains, and failed queries, the data set had 13.6M DNS responses. Applying our *consistency* and *independent verifiability* metrics, we identified 42K manipulated responses (0.3% of all responses) for 1,408 domains, spanning 58 countries (and dependent territories).

Manipulation by Country

Previous work has observed that some countries deploy nationwide DNS censorship technology; therefore, we expect to see groups of resolvers in the same countries, where each group of resolvers manipulates similar sets of domains. Table 1 shows the median percentage of manipulated responses per resolver, aggregated across resolvers in the top censored country.

Which Countries Experience the Most DNS Manipulation?

Resolvers in Iran exhibited the highest degree of manipulation, with a median of 6.02% manipulated responses per Iranian resolver; China follows with a median value of 5.22%. The relative rankings of countries depend on the domains in our input data set.

For example, if sites censored in Iran and China are overrepresented in the CLBL, the overall rankings will skew towards those countries. Creating an unbiased globally representative set of test domains remains an open research problem.

Are There Outliers within Countries?

Yes. Each country shown in Table 1 had at least one resolver that did not manipulate *any* domains. This effect likely results from IP address geolocation inaccuracies. For example, resolvers in Hong Kong (which are not subject to Chinese Internet censorship) were incorrectly labeled by MaxMind as Chinese. Additionally, for countries that rely on individual ISPs to implement government censorship, the actual manifestation of manipulation can vary across ISPs within the country. Localized manipulation by resolver operators in countries with few resolvers can also influence these results. Similarly, most countries had at least one resolver that showed DNS manipulation significantly greater than the median. This again points to localized manipulation, such as corporate networks deploying firewall products that block content unrelated to state-mandated censorship.

Consistency within Countries

Figure 3 shows the DNS manipulation of each domain by the fraction of resolvers *within* a country, for the 10 countries with the greatest (normalized) level of manipulation. Each point represents a domain; the vertical axis represents the fraction of resolvers in that country that manipulated it. Shading shows the density of points for that part of the distribution. We only plot domains that experience blocking by at least one resolver within a given country.

Country	Number Resolvers	Median Manipulation
Iran	122	6.02%
China	62	5.22%
Indonesia	80	0.63%
Greece	26	0.28%
Mongolia	6	0.17%
Iraq	7	0.09%
Bermuda	2	0.04%
Kazakhstan	14	0.04%
Belarus	18	0.04%

Table 1: Top countries by median percent of manipulated responses per resolver

Heterogeneity across a country suggests that different ISPs may implement filtering with different block lists; it might also indicate variability of blocking policies across geographic regions within a country.

Is There Heterogeneity Within Countries?
Yes. For example, one group of domains was manipulated by about 80% of resolvers in Iran, and another group was manipulated by fewer than 10% of resolvers. Similarly, one set of domains in China experienced manipulation by approximately 80% of resolvers, and another set experienced manipulation only about half of the time.

Is There Non-Determinism in Censorship?
Yes. Effects that appear as smearing across the vertical axis, such as for Iran and China, denote instances where individual domains were not manipulated by an identical set of resolvers but rather by an *almost* identical set. These phenomena can arise as the result of censorship systems using DNS *injection*, where a race condition exists between the injected and the original responses. It can also occur if systems under load fail open, or if the censor operates its manipulations in a probabilistic manner.

Is There Geolocation Inaccuracy?
Yes. Upper bounds on the proportion of resolvers within a country performing manipulation suggest IP geolocation errors are common. For example, no domain in China experienced manipulation across more than approximately 85% of resolvers. This is also supported by the frequency of outliers within countries performing no manipulation, as discussed earlier.

Rank	Domain Name	Category	Countries
1	www.*pokerstars.com	Gambling	19
2	betway.com	Gambling	19
3	pornhub.com	Pornography	19
4	youporn.com	Pornography	19
5	xvideos.com	Pornography	19
6	thepiratebay.org	P2P sharing	18
7	thepiratebay.se	P2P sharing	18
8	xhamster.com	Pornography	18
9	www.*partypoker.com	Gambling	17
10	beeg.com	Pornography	17
80	torproject.org	Anon. & cen.	12
181	twitter.com	Twitter	9
250	www.*youtube.com	Google	8
495	www.*citizenlab.org	Freedom expr.	4
606	www.google.com	Google	3
1086	google.com	Google	1

Table 2: Domain names manipulated in the most countries, ordered by number of countries with manipulated responses

Manipulation by Domain
Table 2 highlights which specific domains experienced manipulation in numerous countries, ranked by the number of countries.

Which Domains Are Most Frequently Manipulated?
The two most manipulated domains were both gambling Web sites, each experiencing censorship across 19 different countries. DNS resolutions for pornographic Web sites were similarly manipulated, accounting for the next three most commonly affected domains. Peer-to-peer file sharing sites were also commonly targeted, particularly The Pirate Bay.

Are Commonly Measured Sites Manipulated by the Most Countries?
No. Sites such as The Tor Project, Citizen Lab, Google, and Twitter are common censorship measurement targets. Yet our results show these sites experienced manipulation by significantly fewer countries than others (bottom half of Table 2). The Tor Project DNS domain, manipulated by 12 countries, was the most widely interfered with among anonymity and censorship tools; Citizen Lab experienced manipulation by four countries. Such disparity points to the need for a diverse domain data set.

Rank	Domain Category	Countries
1	Alexa Top 10K	36
2	Freedom of expr.	35
3	P2P file sharing	34
4	Human rights	31
5	Gambling	29
6	Pornography	29
7	Alcohol and drugs	28
8	Anon. & censor.	24
9	Hate speech	22
10	Multimedia sharing	21
20	Google (All)	16
34	Facebook (All)	10
38	Twitter (All)	9

Table 3: Top 10 domain categories, ordered by number of countries with manipulated answers

Manipulation by Category

Table 3 shows the prevalence of manipulation by CLBL categories. We consider a category as manipulated within a country if any resolver within that country manipulated a domain of that category.

Which Types of Domains Are Most Frequently Manipulated?

Domains in the Alexa Top 10K experienced the most manipulation; these domains did not appear in the CLBL, which highlights the importance of measuring both curated lists from domain experts as well as broad samples of popular Web sites. Although no single domain experienced manipulation in more than 19 countries, several categories experienced manipulation in more than 30 countries, indicating that while broad categories appear to be commonly targeted, the specific domains vary country to country.

Are Commonly Measured Sites in the Most Frequently Manipulated Categories?

No. As was the case with domain ranking, common platforms such as Google, Facebook, and Twitter witnessed manipulation across significantly fewer countries than other categories.

Are the Top Manipulated Sites from the Top Manipulated Categories?

No. While eight of the top 10 most frequently manipulated sites were in the Gambling and Pornography categories, those categories ranked 5th and 6th overall when aggregated. This difference highlights the need for measurement studies to consider multiple aggregation metrics when reporting ranked censorship results.

Conclusion

Internet censorship is widespread, dynamic, and continually evolving; understanding the nature of censorship thus requires techniques to perform continuous, large-scale measurements.

Iris supports regular, continuous measurement of DNS manipulation, ultimately facilitating global tracking of DNS-based censorship as it evolves over time. Our first large-scale measurement study using Iris highlights the heterogeneity of DNS manipulation across countries, resolvers, and domains, and demonstrates the potential of operationalizing such measurements for longitudinal analysis.

Acknowledgments

The authors are grateful for the assistance and support of Manos Antonakakis, Randy Bush, Jed Crandall, Zakir Durumeric, and David Fifield. This work was supported in part by National Science Foundation Awards CNS-1237265, CNS-1406041, CNS-1518878, CNS-1518918 CNS-1540066 and CNS-1602399.

References

[1] S. Burnett and N. Feamster, "Encore: Lightweight Measurement of Web Censorship with Cross-Origin Requests," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pp. 653-667: http://bit.ly/2yw5OHZ.

[2] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning," ACM Conference on Computer and Communications Security (CCS '15): https://censys.io/static/censys.pdf.

[3] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-Wide Scanning and Its Security Applications," in *Proceedings of the 22nd USENIX Security Symposium (Security '13)*: http://bit.ly/2wHvyE7.

[4] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall, "Detecting Intentional Packet Drops on the Internet via TCP/IP Side Channels," in *Proceedings of the International Conference on Passive and Active Network Measurement (PAM '14)*, pp. 109-118.

[5] Freedom House, Freedom on the Net, "Silencing the Messenger: Communication Apps under Pressure," 2016: https://freedomhouse.org/report/freedom-net/freedom-net-2016.

[6] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going Wild: Large-Scale Classification of Open DNS Resolvers," ACM Internet Measurement Conference (IMC '15): http://bit.ly/2xoVG3T.

[7] P. Pearce, R. Ensafi, F. Li, N. Feamster, and V. Paxson, "Augur: Internet-Wide Detection of Connectivity Disruptions," IEEE Symposium on Security and Privacy (S&P '17): http://bit.ly/2nlR1cY.

[8] P. Pearce, B. Jones, F. Li, R. Ensafi, N. Feamster, N. Weaver, and V. Paxson, "Global Measurement of DNS Manipulation," in *Proceedings of the 26th USENIX Security Symposium (Security '17)*: http://bit.ly/2xA6Qoy.

[9] A. Razaghpanah, A. Li, A. Filastò, R. Nithyanand, V. Ververis, W. Scott, and P. Gill, "Exploring the Design Space of Longitudinal Censorship Measurement Platforms," Technical Report 1606.01979, ArXiv CoRR, 2016: https://arxiv.org/pdf/1606.01979.pdf.

[10] The Tor Project, "OONI: Open Observatory of Network Interference": https://ooni.torproject.org/.

An End-to-End View of DNSSEC Ecosystem Management

TAEJOONG CHUNG, ROLAND VAN RIJSWIJK-DEIJ, BALAKRISHNAN CHANDRASEKARAN, DAVID CHOFFNES, DAVE LEVIN, BRUCE M. MAGGS, ALAN MISLOVE, AND CHRISTO WILSON

Taejoong Chung is a Postdoctoral Researcher in the College of Computer and Information Science at Northeastern University. His work focuses on Internet security, privacy implications, and big data analysis through large-scale measurements. t.chung@neu.edu

Roland van Rijswijk-Deij is an enthusiastic Internet researcher, specializing in measurement-based research with a focus on DNS, DNSSEC, and network security. Roland has a PhD in computer science from the University of Twente in The Netherlands. He works for SURFnet, the National Research and Education Network in The Netherlands, and as Assistant Professor in the Design and Analysis of Communication Systems group at the University of Twente. r.m.vanrijswijk@utwente.nl

Balakrishnan Chandrasekaran is a Senior Research Scientist at Technische Universität Berlin. He received his PhD in computer science from Duke University. His research focuses on making the Internet faster, reliable, and more secure, and his work spans network measurements and mapping, network security, and software-defined networking. balac@inet.tu-berlin.de

The Domain Name System (DNS) provides name resolution for the Internet, and DNS's Security Extensions (DNSSEC) allow clients and resolvers to verify that DNS responses have not been forged. DNSSEC can operate securely only if each of its principals performs its management tasks correctly: authoritative name servers must generate and publish their keys and signatures, domains that support DNSSEC must be signed with their parent's keys, and resolvers must actually validate the chain of signatures. We perform the first large-scale measurement study into how well DNSSEC's PKI is managed, studying the behavior of domain operators, registrars, and resolvers. Our investigation reveals pervasive mismanagement of the DNSSEC infrastructure: only 1% of the .com, .org, and .net domains attempt to deploy DNSSEC; many popular registrars that support DNSSEC fail to publish all relevant records required for validation; and only 12% of resolvers that request DNSSEC records actually attempt to validate them.

The Domain Name System (DNS) is the Internet's equivalent of the "yellow pages": it translates human-readable domain names to machine-friendly Internet Protocol (IP) addresses. Unfortunately, the original DNS protocol did not include any security mechanisms. This lack of security allows an adversary to forge DNS records, and such attacks can have significant effects on end users, who may end up unknowingly communicating with malicious servers.

To address these problems, the DNS Security Extensions (DNSSEC) were introduced nearly two decades ago. At its core, DNSSEC is a hierarchical public key infrastructure (PKI) that largely mirrors the DNS hierarchy and is rooted at the DNS root zone. To enable DNSSEC, the owner of a domain signs its DNS records (using its private key) and publishes the signatures along with its public key; this public key is then signed by its parent domain, and so on up to the DNS root zone, resulting in a *chain of trust*. As of early 2017, more than 90% of top-level domains (TLDs), such as .com, and 47% of country-code TLDs (ccTLDs), such as .nl, are DNSSEC-enabled [4, 8]. DNS resolvers that perform recursive DNS lookups on behalf of end users validate DNSSEC signatures in order to ensure that the response to a query they handle is authentic and was not modified in flight. These so-called *validating resolvers* perform signature verification along the chain of trust, from the signature on the record that was requested all the way to the top of the PKI at the root of the DNS. But like any PKI, DNSSEC can only function correctly when all principals—every signatory from root to leaf and the resolver validating the signatures—fulfill their respective responsibilities. Unfortunately, DNSSEC is complex, creating many opportunities for mismanagement.

On the authoritative server side, a single error such as a weak key or an expired signature can weaken or completely compromise the integrity of a large number of domains. *On the resolver side,* mismanaged or buggy DNS resolvers can obviate all server-side efforts by simply failing to catch invalid or missing signatures.

David Choffnes is an Assistant Professor in the College of Computer and Information Science and a member of the Cybersecurity and Privacy Institute at Northeastern University. His research is primarily in the areas of distributed systems and networking, focusing on mobile systems, privacy, and security. His research has been supported by the NSF, DHS, Comcast Innovation Fund, Google Research Awards, the Data Transparency Lab, M-Lab, and a Computing Innovations Fellowship. choffnes@ccs.neu.edu

Dave Levin is an Assistant Professor of Computer Science and Chair of the Computer Science Honors program at the University of Maryland, from which he also received his BS and PhD. His research combines measurement and systems building to improve the security of the Internet, including the Web's public key infrastructure, DNS, and censorship avoidance. dml@cs.umd.edu

Bruce Maggs is the Pelham Wilder Professor of Computer Science at Duke University and Vice President for Research at Akamai Technologies. His research interests focus on distributed systems, including content delivery networks, computer networks, and computer and network security. bmm@cs.duke.edu

In this article, we present a comprehensive study of the entire DNSSEC ecosystem—encompassing signers, authoritative name servers, registrars, and validating DNS resolvers—to understand how DNSSEC is (mis)managed today. To study server-side behavior, our work relies on 21 months of daily snapshots of DNSSEC records for *all* signed .com, .net, and .org second-level domains. To study resolver-side behavior, we purchased domains from the most popular 20 registrars (responsible for 54.3% of all .com, .net, and .org domains), as well as the 10 registrars that operate the most domains with "DNSKEY"s (covering 84.6% of such domains in .com, .net, and .org). To study client-side behavior, we leverage the Luminati HTTP proxy service, which allows us to perform repeated, controlled tests from 403,355 end hosts and their 59,513 distinct DNS resolvers around the world.

Our analysis reveals troubling, persistent mismanagement in the DNSSEC PKI:

- First, we find that nearly *one-third* of DNSSEC-enabled domains produce records that *cannot be validated* due to missing or incorrect records. The vast majority of these missing records are due to registrars that host many domains but fail to publish the correct records for domains they manage.
- Second, we find that registrar support for DNSSEC varies widely. Among the top 20 registrars, only three support DNSSEC when the registrar runs the authoritative DNS server (referred to as being the *DNS operator*); only one does so by default, and then only for some of its more expensive plans. Moreover, not all of the registrars we study support DNSSEC when the domain owner is the DNS operator. Of those that do, many require cumbersome and insecure steps for domain owners to deploy DNSSEC, such as requiring that domain information be sent over insecure email channels.
- Third, we find that although 58% of observed resolvers request DNSSEC records during their queries, only 12% of them actually validate the records. This means that the majority of resolvers pay the overhead to download DNS records for DNSSEC, while not reaping the security benefits.

In summary, our results paint a distressing picture of widespread mismanagement of keys and DNSSEC records that violate best practices in some cases and completely defeat the security guarantees of DNSSEC in others. On a more positive note, our findings demonstrate several areas of improvement where management of the DNSSEC PKI can be automated and audited. To this end, we have publicly released all of our analysis code and data (where possible) to the research community at https://securepki.org, thereby allowing other researchers and administrators to reproduce and extend our work.

Background

DNS

The Domain Name System (DNS) is based on *records* that map *domain names* (e.g., "example.com") to Internet Protocol (IP) addresses (e.g., "10.0.0.1"). DNS is a distributed system, and there are three primary kinds of organizations involved in the domain name registration process:

- *Registries* are organizations that manage top-level domains (TLDs). They maintain their TLD *zone file* (the list of all registered names in that TLD). For example, Verisign serves as the registry for .com.
- *Registrars* are organizations that sell domains to the public. Because they are accredited by ICANN, they can directly access the registry, which enables them to process new registrations.
- *DNS operators* are organizations that run *authoritative* DNS servers. Each domain has a DNS operator; the most common cases are (1) the domain owner asks their registrar to run the authoritative DNS server (registrar DNS operator), or (2) the domain owner runs their own authoritative DNS server (owner DNS operator).

An End-to-End View of DNSSEC Ecosystem Management

Alan Mislove is an Associate Professor and Associate Dean and Director of Undergraduate Programs at the College of Computer and Information Science at Northeastern University, which he joined in 2009. He received his BA, MS, and PhD in computer science from Rice University. Prof. Mislove's research focuses on the security and privacy implications of today's distributed systems, often centered around large-scale measurement and analysis. amislove@ccs.neu.edu

Christo Wilson is an Assistant Professor in the College of Computer and Information Science at Northeastern University and a member of the Cybersecurity and Privacy Institute at Northeastern. His work focuses on Web security, privacy, and algorithmic transparency. His work is funded by the NSF, the Russell Sage Foundation, the European Commission, and the Data Transparency Lab. cbw@ccs.neu.edu

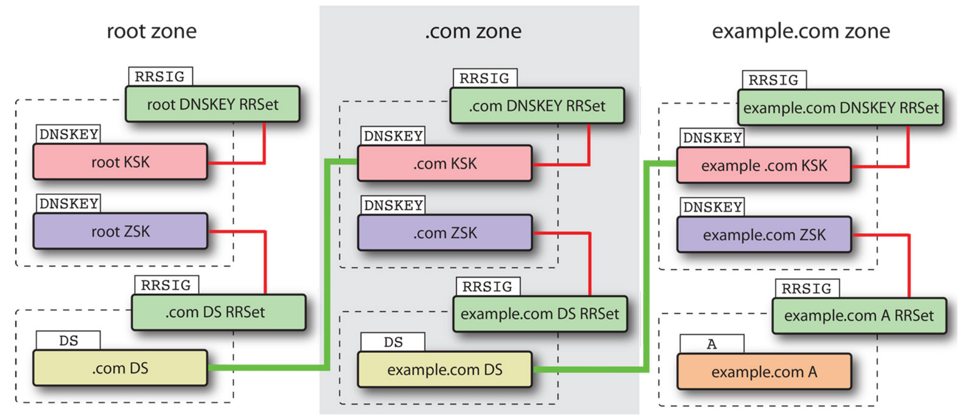


Figure 1: Overview of DNSSEC records necessary to validate example.com's "A" record. Each RRSIG is the signature of a record set (dashed lines) verified with a DNSKEY (thinner solid lines). Each DS record is the hash of a child zone's KSK, or key-signing key (thicker solid lines).

Whenever a registrar sells a domain name, it must insert an "NS" (name server) record for the new domain into the registry's TLD zone file; the "NS" record contains the identity of the authoritative DNS server (i.e., the DNS operator).

DNSSEC

Unfortunately, the original DNS protocol did not include authenticity mechanisms, allowing an adversary to forge DNS responses. The DNS Security Extensions (DNSSEC) are designed to address this vulnerability. DNSSEC provides integrity for DNS records using three primary record types:

- "DNSKEY" records are public keys used to validate DNS records in DNSSEC.
- "RRSIG" (Resource Record Signature) records are cryptographic signatures of other records. Each RRSIG is created using the private key that matches a DNSKEY; all records need to carry signatures to ensure that they are not forged.
- "DS" (Delegation Signer) records are essentially hashes of DNSKEYs. These records must be uploaded to the parent zone, where they are signed by the parent's DNSKEY.

Resolvers

Most Internet hosts are configured to use a local DNS *resolver*, which looks up domain names for them. The resolver iteratively determines the authoritative DNS server for a domain, obtains the requested record, and forwards it back to the requesting host. If the resolver supports DNSSEC, it will also fetch all DNSSEC records (DNSKEYs and RRSIGs) and validate them. Finally, the resolver returns the (validated) record back to the requesting host.

A resolver indicates that it would like to receive DNSSEC records by setting the "DO"(DNSSEC OK) bit in its DNS requests. Then the responding authoritative DNS server will include the RRSIGs corresponding to the record type of the request in its response. Once it receives the RRSIGs, the resolver can then fetch the necessary DNSKEYs and DS records to validate the response.

Validating a DNSSEC Record

All DNSSEC-aware resolvers must be provided with the root zone's key. There is a logical chain of DNSKEYs, starting from the root zone's key through the desired zone's DNSKEY record. Once a domain's DNSKEY has been authenticated, the record in question can be validated using this key and the record's RRSIG. Figure 1 shows example records and how they are related.

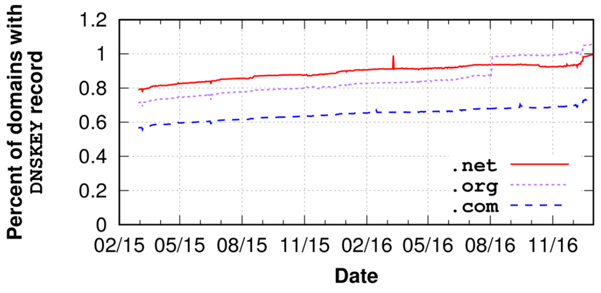


Figure 2: The percentage of all .com, .org, and .net second-level domains that have a DNSKEY record. Between 0.75% and 1.0% of all domains publish a DNSKEY record in our latest snapshot.

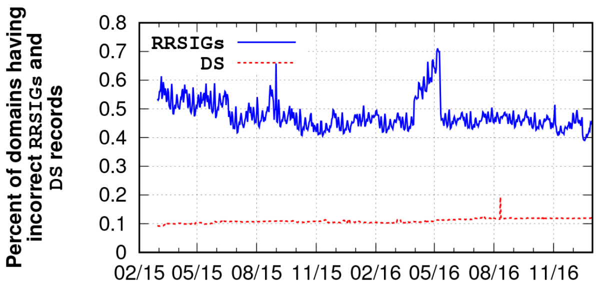


Figure 3: The percent of signed domains for which the RRSIG signatures or DS records are invalid

Uploading DS Records

If a DNS operator wishes to support DNSSEC, a DS record for the domain must be uploaded to the registry (along with the NS record) in order to establish a chain of trust. However, only registrars can upload DS records to the registry. Thus, if the domain's DNS operator is the registrar, the operator can simply upload the DS record by directly accessing the registry. Unfortunately, if the domain's DNS operator is the owner, the situation is more complicated since the registrar does not know the DS record. To this end, a registrar may provide customers with a Web-based interface to submit DS records, or may allow customers to transmit DS records via an out-of-band mechanism such as by email or telephone. Moreover, if a registrar does not support any methods for customers to upload DS records, the domain *cannot* support DNSSEC since it will have a broken chain of trust due to the missing DS record.

Authoritative Name Servers

We begin our analysis of the DNSSEC PKI by focusing on the deployment and management of DNSSEC records by domains and how this has changed over time.

Data Sets

This section describes a large-scale, longitudinal, and detailed study of DNSSEC adoption and deployment at authoritative name servers. To this end, we use data from OpenINTEL [7, 9] concerning domains listed in zone files for the .com, .net, and .org TLDs; together, these contain approximately 150M domains and cover 64% of the Alexa Top-1M (and 75% of the Alexa Top-1K sites). OpenINTEL collects daily snapshots of key DNS records for all of these 150M domains. For this study, we used the NS, DS, SOA, DNSKEY, and RRSIG records that OpenINTEL collected for .com, .net, and .org domains. These daily snapshots span 21 months (between March 1, 2015 and December 31, 2016).

DNSSEC Prevalence

We begin by examining how support for DNSSEC has evolved over time. Specifically, we focus on the number of second-level domains (e.g., amazon.com) that publish at least one DNSKEY record; we refer to these as *signed domains*. Note that having a DNSKEY record published does not by itself imply that the domain has correctly deployed DNSSEC; there could be other missing records or invalid signatures; rather, this indicates that the domain *attempted* to deploy DNSSEC.

Figure 2 plots the fraction of .com, .net, and .org second-level domains that publish at least one DNSKEY record. One key observation is that DNSSEC deployment is rare: between 0.6% (.com) and 1.0% (.org) of domains have DNSKEY records published in our latest snapshot. The fraction of domains that have DNSKEYs is, however, steadily growing. Because the trends of deployment and growth for each TLD are similar, for the remainder of this section, we combine the TLDs into a single data set; breakdowns into different TLDs are available in our recent paper [2].

Incorrect Records

Next, we study whether the DS records and RRSIGs published by signed domains are *correct*: the DS record should match the hash of the DNSKEY, and the RRSIGs should not be expired and should validate against the DNSKEYs. Figure 3 presents the results. We find that the results are largely positive. Almost 99.9% of signed domains have DS records that match their DNSKEY. (The spike that occurred in August 2016 was caused by domains hosted by one authoritative name server, transip.net. This name server suddenly changed DNSKEYs for over 400 domains without switching the DS record, and the problem was corrected the following day.) Similarly, we find that over 99.5% of signed domains have correct RRSIGs and that the majority of the incorrect RRSIGs are due to signature expirations (RRSIGs, unlike DS records, have an expiry timestamp built in).

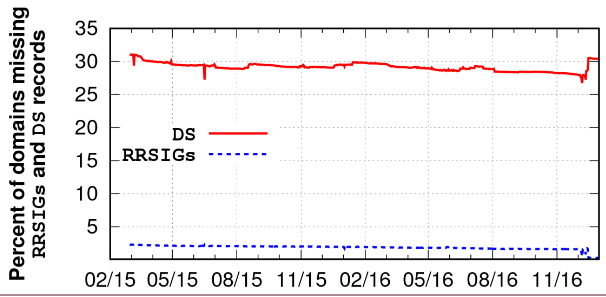


Figure 4: The percentage of signed domains that fail to publish a DS record in the parent zone and RRSIG for SOA and DNSKEY

Name Servers	Signed	w/DS	Ratio
*.ovh.net	316,960	315,204	99.45%
*.loppia.se	131,726	1	0.00%
*.hyp.net	94,084	93,946	99.85%
*.transip.net	91,103	91,009	99.90%
*.domainmonster.com	60,425	4	0.01%
*.anycast.me	52,381	51,403	98.13%
*.transip.nl	47,007	46,971	99.92%
*.binero.se	44,650	17,099	38.30%
*.ns.cloudflare.com	28,938	17,483	60.42%
*.is.nl	15,738	11	0.07%
*.pcextreme.nl	14,967	14,801	98.89%
*.webhostingserver.nl	14,806	10,655	71.96%
*.registrar-servers.com	13,115	11,463	87.40%
*.nl	12,738	12,674	99.50%
*.citynetwork.se	11,660	13	0.11%

Table 1: Table showing the 15 most popular common domains listed in NS records for authoritative name servers, the total number of signed domains, and the number of domains with a DS record for our latest snapshot (December 31, 2016). The shaded rows represent registrars that fail to publish DS records for nearly all of their domains.

Missing Records

We now examine whether signed domains are publishing all necessary DNSSEC records. Recall that properly deploying DNSSEC for a domain means that it must have a DS record in the parent zone, DNSKEY records, and RRSIG records for every published record type.

Figure 4 shows the percentage of domains that have DNSKEYs but are missing DS or RRSIG records. We can immediately observe that while a surprisingly low fraction of signed domains are missing RRSIGs (2%, on average), between 28%–32% of signed domains do not have a DS record, meaning they cannot be validated.

Recall that, unlike other DNSSEC record types, DS records are published in the *parent* zone (e.g., .com), along with the domain's NS record. Thus, correctly installing a DS record often requires use of an out-of-band channel, where the administrator contacts its registrar and requests that the registrar adds a DS record. To shed light on why so many domains fail to deploy DS records, we group domains by authoritative name servers (i.e., the DNS operator) to see if certain DNS operators are behind the failures. Table 1 shows the results for the 15 most common domains listed in NS records for authoritative name servers, which cover 83% of the signed domains we study. We find a highly skewed distribution, with most of the name servers publishing DS records for almost all signed domains, but with four failing to upload a DS record for nearly all of their domains. For example, Loopia (a Swedish hosting provider) is authoritative for more than 131,000 domains that publish DNSKEYs, but only *one* of these domains actually uploads a DS record, which is invalid.

Registrars

Having observed that DNSSEC is supported by only 1% of .com, .net, and .org domains, and that over 30% of those domains that try to support DNSSEC fail to do so correctly, we now turn to examine the role that registrars play. To do so, we register domains ourselves and attempt to deploy DNSSEC, both with the registrar as the DNS operator and with ourselves as the DNS operator. We focus on the 31 most popular DNS operators across our data sets, which collectively cover 54.3% of .com, .org, and .net domains in the TLD zone files. Table 2 summarizes the results of this experiment. We make a number of observations below.

Registrar as DNS Operator

We first focus on what happens when we use the registrar as the DNS operator for our domain. Surprisingly, only three registrars (GoDaddy, NameCheap, and OVH) out of the 20 we studied support DNSSEC *at all* when they are the DNS operator. This situation is unfortunate because these cases present an easy path to DNSSEC deployment, since the registrar has full control over the domain and could create DNSKEYs, RRSIGs, and upload DS records all on its own. Even more alarming, the three registrars that do support DNSSEC when they are the DNS operator only do so for some of their DNS plans, and *only* NameCheap enables DNSSEC by default. The other two registrars that support DNSSEC also have different policies: GoDaddy provides DNSSEC as a premium package (at a cost of $35 per year), while OVH provides DNSSEC for free but *only* if the customer explicitly opts in. From our December 31, 2016 snapshot, we observe that 25.9% of domains from OVH, 0.59% of domains from NameCheap, and 0.02% of domains from GoDaddy deploy DNSSEC, suggesting that the low DNSSEC adoption rates may be heavily influenced by default options and cost.

Registrar	Domains		Registrar DNS operator		Owner DNS operator				
	All	with	DNSSEC	DNSSEC	DNSSEC	DS upload		DS Validation	
		DNSKEY	default	opt-in	support	Web	Email	DNSKEY	Email
GoDaddy	37,652,477	8,139	✗	●	●	●	-	✗	-
Alibaba	4,292,138	3	✗	✗	✗	-	-	-	-
1AND1	3,802,824	0	✗	✗	✗	-	-	-	-
Network Solution	2,534,673	0	✗	✗	✗	-	-	-	-
eNom	2,525,828	10	✗	✗	●	✗	●	✗	●
Bluehost	2,066,503	0	✗	✗	✗	-	-	-	-
NameCheap	1,963,717	13,232	▲	-	●	●	-	✗	-
WIX	1,887,139	0	✗	✗	-	-	-	-	-
HostGator	1,849,735	0	✗	✗	✗	-	-	-	-
NameBright	1,823,823	0	✗	✗	●	✗	●	✗	▲
register.com	1,311,969	0	✗	✗	✗	-	-	-	-
OVH	1,228,578	319,580	✗	●	●	●	-	●	-
DreamHost	1,117,902	0	✗	✗	●	✗	●	●	▲
WordPress	888,174	3	✗	✗	✗	-	-	-	-
Amazon	865,065	0	✗	✗	●	●	-	▲	-
Xinnet	836,293	0	✗	✗	✗	-	-	-	-
Google	813,945	1,945	✗	✗	●	●	-	✗	-
123-reg	720,435	1	✗	✗	●	●	-	✗	-
Yahoo	690,823	0	✗	✗	✗	-	-	-	-
Rightside	663,616	0	✗	✗	●	●	-	✗	-

Table 2: Table showing the results of our study of registering domains using the 20 registrars among the top 29 DNS operators. The other nine DNS operators are parking services or malware domains. Only three of the 20 support DNSSEC for domains they manage, and only one of them provides DNSSEC by default for these domains (NameCheap only supports DNSSEC by default for certain plans, hence the ▲s [6]). Only 11 of the registrars support DNSSEC for external name servers, eight providing Web-based forms for uploading DS records, and three requiring emails with DS records; only two of these actually validate the provided DS records. Of the three that require emails, two of them do not verify the validity of the incoming email (hence the ▲s).

Owner as DNS operator

Next, we explore how registrars support DNSSEC if the owner acts as the DNS operator (e.g., by hosting their own name server). We find that only 10 of the 20 registrars support DNSSEC for such domains.

Interestingly, only three of the 10 registrars present a DS upload menu on their Web interface when a user switches to an external name server; others use mechanisms such as support tickets or require emails to allow customers to provide DS records. Using email is particularly distressing, since communicating DS records over email opens up security vulnerabilities due to the insecurity of email communication.

DS Record Validation

We now turn our attention to see whether these registrars validate submitted DS records. While registrars are not *required* to validate DS records, they are best positioned to help their customers deploy DNSSEC. We first checked whether the registrars validate the uploaded DS record to ensure it is the hash of the domain's DNSKEY; only two registrars correctly validated the DS record before accepting it. The remaining registrars all allowed us to publish arbitrary data as DS records. We then tested whether the registrars that require emailed DS records would accept an updated DS record without confirming the update. We found that two of the three registrars that require emailed DS records did not attempt to verify the email, meaning an attacker who wished to take control of a victim domain could do so by forging an email to these registrars. We have contacted these two registrars to inform them of this security vulnerability.

DNS Resolvers

Even if domains properly manage their DNSSEC records, end hosts do not enjoy the benefits of DNSSEC unless their DNS resolver requests and validates these records properly. We now examine the DNSSEC behavior of resolvers.

Methodology

A challenge when studying the behavior of resolvers is that most will respond only to local clients (i.e., most are not open resolvers). To address this limitation, we use the Luminati proxy network [1] to issue DNS requests. Luminati is composed of nodes that act as HTTP proxies, which allow us to (1) select the country where the node (managed by Luminati) is located and (2) route HTTP traffic via the node. The node then makes a DNS request for the domain we specify, makes the HTTP request, and returns the response back via the Luminati proxy network.

For this section, we only focus on (1) nodes that are configured with a single resolver and (2) resolvers that we were able to measure with at least 10 different nodes; this represents total 7,599 resolvers covering 328,666 total nodes in 3,582 autonomous systems (ASes). See [1, 2] for more details on this service and the methodology we used for this measurement.

Domain Configuration

For these experiments, we built an authoritative DNS server and Web server for a testbed domain under our control. Our testbed domain (a second-level domain) fully supports DNSSEC functionality with a chain of trust by uploading its DS record to the .com zone.

One of our goals is to examine whether DNSSEC resolvers properly validate DNSSEC records. To do so, we configured our DNS server with 10 different subdomains, each of which simulates a different kind of DNSSEC misconfiguration, along with a single *valid* zone. These misconfigurations include missing, incorrect, and expired RRSIGs, missing DNSKEYs, incorrect DS records, etc.

Results

Of the 7,599 resolvers we examined, we found that 4,427 (58.3%) of them send requests with the DO bit set, suggesting that a majority of resolvers support DNSSEC. We refer to this set of resolvers that request DNSSEC records as *DNSSEC-aware resolvers*. Setting the DO bit by itself, however, does not indicate that the resolvers actually *validate* the DNSSEC responses they receive. To test for proper validation, we look at whether each HTTP request made via a node was successful; because all but one of our DNSSEC records are misconfigured, we would expect all of our HTTP requests (except for those to a single valid domain) to fail validation.

Incorrectly Validating Resolvers

We found that 3,635 of the DNSSEC-aware resolvers (82.1%) from 301 ASes consistently fail to validate the DNSSEC responses, even though they issue the DNS requests with the DO bit set; these resolvers cover 149,373 (78.0%) of the nodes with DNSSEC-aware resolvers. This is especially surprising, as these resolvers all pay the overhead for DNSSEC responses but do not bother to reap DNSSEC's benefits by validating the results they receive.

Country	Hosting ISP	Resolvers	Nodes
Indonesia	PT Telekomunikasi	1,319	2,695
U.S.	Level 3 Communications	522	79,303
U.S.	Time Warner Cable Internet	148	1,133
Germany	Deutsche Telekom AG	104	2,682
Canada	Bell Canada	89	1,120
U.K.	TalkTalk Communications	76	878
U.K.	Sky UK Limited	74	1,535
U.S.	Frontier Communications	63	241
China	China Telecom	56	344
Canada	Rogers Cable Communications	49	1,250
Spain	Telefonica de Espana	48	1,982
U.S.	Charter Communications	46	355
Austria	Liberty Global Operations	40	10,554
U.S.	SoftLayer Technologies	37	2,559
Czech	Avast Software s.r.o.	33	2,731

Table 3: The top 15 ISPs in terms of the number of DNS resolvers that do not validate our DNSSEC response. Level 3 (shaded) has 522 resolvers that do not validate the DNSSEC response, while six do (not shown).

Table 3 shows the top 15 ASes where we observe resolvers that set the bit but do not validate DNSSEC responses; we can immediately observe that these networks include large, popular ISPs in the U.S., the U.K., Canada, and Germany.

Correctly Validating Resolvers

Only 543 of the DNSSEC-aware resolvers (12.2%) from 129 ASes consistently correctly validate DNSSEC responses; these resolvers cover 31,811 (16.6%) of the nodes covered by DNSSEC-aware resolvers. We found surprisingly few large ASes that validate DNSSEC responses; the largest ones include Comcast (U.S.), Orange (Poland), Bahnhof Internet AB (Sweden), Free SAS (France), and EarthLink (Iraq). Interestingly, we found that all validating resolvers successfully validate all misconfigured scenarios; we did not find any resolvers that failed some of our misconfiguration tests but passed others. This is in contrast to client behavior for other PKIs, such as the Web [5], where browsers pass different subsets of validation tests.

Conclusion

Taken together, our results indicate there are a number of steps that the various DNS entities can take to spur greater adoption of DNSSEC.

- First, DNS resolver software should enable DNSSEC validation by default; many popular implementations request DNSSEC records by default, but then completely ignore them.
- Second, registrars should allow all customers to enable DNSSEC if they wish, and should move towards a standard of DNSSEC-by-default; today, only one registrar among the top 20 has this policy.
- Third, registries should support the "CDS" and "CDNSKEY" proposals [10], which allow domain owners to directly communicate DS records to the registry; unfortunately, we know of very few registries that support CDS and CDNSKEY today.
- Fourth, until CDS and CDNSKEY are fully supported, registrars should work to make the process of uploading DS records easier and more secure.

We also encourage interested readers to read our recent papers on DNSSEC [2, 3], which collectively explore this topic in greater detail.

References

[1] T. Chung, D. Choffnes, and A. Mislove, "Tunneling for Transparency: A Large-Scale Analysis of End-to-End Violations in the Internet," IMC, 2016: https://mislove.org/publications/Luminati-IMC.pdf.

[2] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "A Longitudinal, End-to-End View of the DNSSEC Ecosystem," in *Proceedings of the 26th USENIX Security Symposium (Security '17)*: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-chung.pdf.

[3] T. Chung, R. van Rijswijk-Deij, D. Choffnes, A. Mislove, C. Wilson, D. Levin, and B. M. Maggs, "Understanding the Role of Registrars in DNSSEC Deployment," IMC, 2017.

[4] ICANN TLD DNSSEC Report: http://stats.research.icann.org/dns/tld_report.

[5] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, A. Schulman, and C. Wilson, "An End-to-End Measurement of Certificate Revocation in the Web's PKI," IMC, 2015: https://www.cs.umd.edu/~dml/papers/revocations_imc15.pdf.

[6] Name servers and TLDs supported/unsupported by DNSSEC: http://bit.ly/2fbNjAp.

[7] OpenINTEL: https://www.openintel.nl/.

[8] State of DNSSEC Deployment 2016: http://bit.ly/2ye4vfX.

[9] R. van Rijswijk-Deij, M. Jonker, A. Sperotto, and A. Pras, "A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, 2016.

[10] P. Wouters and O. Gudmundsson, "Managing DS Records from the Parent via CDS/CDNSKEY," RFC 8078, IETF, 2017: https://datatracker.ietf.org/doc/rfc8078/.

Securing the Internet, One HTTP 200 OK at a Time

WILFRIED MAYER, KATHARINA KROMBHOLZ, MARTIN SCHMIEDECKER, AND EDGAR WEIPPL

Wilfried Mayer is a Researcher at SBA Research and a PhD student at Technische Universität Wien. His research interests are mainly empirical security measurements. He also likes to make the Internet a better place. wmayer@sba-research.org

Katharina Krombholz is Senior Researcher at SBA Research. Her research focuses on usable security, privacy, and digital forensics. She is currently interested in usable security aspects of crypto deployments, IoT, and usable security research methodology. kkrombholz@sba-research.org

Martin Schmiedecker is Senior Researcher at SBA Research. His research interests include everything related to digital forensics, online privacy, and applied security. mschmiedecker@sba-research.org or @Fr333k

Edgar Weippl is Research Director at SBA Research and Associate Professor (Privatdozent) at the Technische Universität Wien. His research focuses on applied concepts of IT security and e-learning. eweippl@sba-research.org

HTTPS is the most commonly used cryptographic protocol on the Internet. It protects communication content and provides endpoint authenticity at scale. However, deploying HTTPS in a truly secure fashion can be a challenging task even for experienced admins. To explore why this is the case and how these challenges can be fixed in order to support an even wider adoption, we conducted a user study, which was presented at USENIX Security 2017.

Targeting the Long Tail

Nowadays, major online services provide TLS encrypted communication. But is the Web site of your local sushi restaurant secured with HTTPS? Because even your local sushi place needs HTTPS! We need HTTPS as the new standard in the Internet for all Web sites and to finally deprecate unencrypted HTTP. But often the opposite is stated. Even at USENIX Security, somebody in the audience questioned the need for HTTPS for small businesses or static content. The answer was simple: more confidentiality, authenticity, and integrity make the Web a more secure place. We know that the upgrade to HTTPS is a way, and a quite promising way, to improve the security of the entire Internet. This improvement is strongly needed, so let's upgrade to HTTPS.

Shifting to HTTPS is not just the problem of single service providers, and the process of enabling it is not just a problem of some individuals. It is an enormous challenge for all of us. Developers, administrators, and security researchers are working on it, and the situation is improving steadily. We see progress, as Felt et al. showed in their recent work [1]. The number of HTTPS page loads is rising and is now exceeding 50% of strict page loads in Firefox telemetry, and there are similar results with Google Chrome statistics. But their work also identifies weak spots. HTTPS usage on Android devices, the deployments in eastern Asia, and the long tail of Web sites are lagging behind. With our study [2], we focus on this long tail of Web sites that are not supporting HTTPS in contrast to major service providers. These companies—ranging from medium enterprises to your local sushi restaurant—do not have highly specialized security experts in charge, actively checking their Web sites for improvements. Normal IT departments, single administrators, and "IT guys" are solving problems here.

And as we already speculate from anecdotes and personal experience, it is not easy to deploy HTTPS in a secure yet compatible fashion. You probably remember your first time, and maybe your last time, setting up HTTPS. Even after initial deployment, the sheer complexity of keeping it secure can overburden experienced admins. Back in 2015, Yan Zhu created a video of knowledgeable members of EFF, who had never set up HTTPS before, being unable to deploy it within a limited amount of time [3].

Enough anecdotes and speculations. We decided to conduct a user study to analyze these problems. We did this without automated tooling provided by Let's Encrypt and chose the traditional approach of using a non-automated CA to issue certificates.

User Study

For the user study, we conducted a series of lab experiments with 28 participants. We recruited students with expert knowledge in the field of security and privacy-enhancing protocols at our university who fulfilled the criteria to potentially work as an administrator or were actually working as administrators. The participants were invited to the lab where they were briefed about the purpose of our study. They assumed the role of administrator of an SME who is in charge of securing the communication to an Apache Web server with HTTPS in order to pass a security audit.

We prepared and implemented a fictive Certificate Authority (CA) in order to facilitate the process of getting a valid certificate and to remove any bias introduced by the procedures from a certain CA. The fictive CA was available through a simple Web interface and required the submission of a valid CSR (certificate signing request) for issuing a valid certificate. The user interface was very simplistic, and the browser on the local machine already trusted our CA. We opted for this study setting because we wanted to focus solely on the actual deployment process instead of on the interaction with a CA. There was no existing TLS configuration on the system—hence the participants had to start a new configuration from scratch. We chose Apache for our experimental setup because Apache maintains a clear lead regarding in-usage share statistics.

We instructed the participants to make the configuration as secure as possible, whereas the assignment did not contain any specific security requirements. In order to collect data, we used a think-aloud protocol. While the participants were working on the task, they articulated their thoughts while an experimenter seated next to them observed their work and took notes. We refrained from video recording due to the results from our pre-test during which we filmed the sessions and noticed a severe impact on the participants' behavior. The participants from the pre-study also explicitly reported that they perceived the cameras as disruptive and distracting, even though the cameras were placed in a discreet way. In addition to the notes from the observation, we captured the bash and browser history and the final configuration files. After completing the task, the participants were asked to fill out a short questionnaire with closed- and open-ended questions that covered basic demographics, previous security experience in industry, and reflections on the experiment.

As a result, we had a collection of both qualitative and quantitative data that was further used for analysis. For a qualitative analysis of the observation protocols, we performed a series of iterative coding which is often used in usable security research to develop models and theories from qualitative data. To evaluate the (mostly) quantitative data acquired via the bash/browser history and Apache log files, we applied metrics and measures to evaluate the quality of the resulting configuration.

Results

For the security evaluation, we based our evaluation criteria on Qualys' SSL Test [4]. We consider this rating scheme a useful benchmark to assess the quality of a TLS configuration based on the state-of-the-art recommendations from various RFCs and with respect to the most recently discovered vulnerabilities and attacks in the protocol.

The rating of the evaluation criteria is expressed with grades from A to F and composed out of three independent values: protocol support (30%), key exchange (30%), and cipher strength (40%). Some properties, e.g., support for the RC4 cipher, cap the overall grade. Only four participants managed to deploy an A grade TLS configuration. B was the most commonly awarded grade (15 out of 28). Four participants did not manage to deploy a valid TLS configuration in the given time.

Our qualitative analysis of the think-aloud protocols from our lab study yielded a process model for a successful TLS configuration. All participants who managed to deploy a valid configuration in the given time can be mapped to the stages presented in this model. The four participants who did not manage to deploy TLS in the given time significantly deviate from this model.

We divide the steps from our model into two phases, a setup phase and a hardening phase. The setup phase refers to a set of tasks to get a basic TLS configuration, i.e., the service is reachable via HTTPS if requested. The hardening phase comprises all necessary tasks to get a configuration that is widely considered secure with respect to the metrics defined by the Qualys SSL Server Rating Guide [5]. Participants who achieved at least a basic configuration successfully completed all steps of the setup phase, while better-graded configurations completed some steps from the hardening phase as well. We identified iterative (tool-supported) security testing as a key element for a successful hardening phase since the participants relied on external sources to evaluate the quality of their configuration.

Usability Challenges

With these results, we identified several usability challenges:

- First, searching for information and finding the right workflow. Our participants visited a high number of Web sites and used multiple sources of information. The average number of visited Web sites was 60, and the most visited pages were the Ubuntu wiki and the official Apache documentation. We found that the participants jumped between sources that were not compatible to each other and could not assess the correctness of the used sources. This included outdated recommendations as well as incomplete tutorials that only covered, for example, the setup phase.

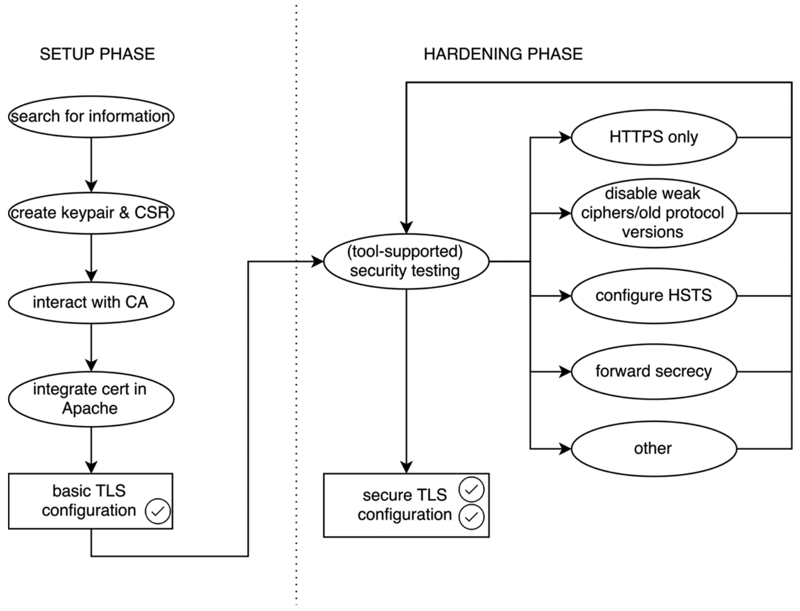


Figure 1: Schematic representation of a successful workflow

- Second, creating a Certificate Signing Request (CSR). The creation of a key pair and the CSR is part of the setup phase. We found that many participants had problems understanding the concept and struggled manually creating the CSR correctly on the first try.
- Third, choosing the appropriate ciphersuites. Ciphersuites define the underlying cryptographic primitives used. A sane configuration defines a limited set of ciphersuites with strong authentication and encryption. This is enabled via one specific Apache directive, "SSLCipherSuite," with the correct values. However, a deep understanding of the underlying algorithms is necessary in order to make an informed decision. All participants trusted online sources because of their missing knowledge, which implies that the quality of these sources is crucial.
- Fourth, strict HTTPS. After finishing a valid HTTPS configuration, most participants tried to enforce HTTPS via redirects and HTTP Strict Transport Security (HSTS) as the first step of the hardening phase. Most participants were initially confused with the default HTTP response when they entered the URL without protocol prefix. They spent a significant amount of time configuring this step correctly.
- Fifth, multiple configuration files. All but six participants struggled with the configuration file structure, regardless of their experience with Apache. Several participants did not understand how to enable the SSL module or where to configure the entry "SSLEngineOn."
- Last, finding the right balance between security and compatibility. In our scenario we didn't specify which level of security the participants should deploy but stated they should make it *as secure as possible*. About 15 of the participants expressed concerns regarding compatibility when configuring SSL/TLS versions, but the majority opted for the more secure options.

Our results reveal that these usability challenges are a serious issue to work on, and that they are the main reason for weak configurations. Our results also show that there is a high demand for improved tool support of the configuration process and more secure default configurations. This would prevent administrators from dealing with mechanisms they cannot fully understand. Fortunately, there are already tools out there. The impact of these tools ranges from generating sane configurations, to comprehensively changing the TLS ecosystem as a whole. We discuss four tools that are all tackling these usability challenges.

Tool Support

One of the projects with the biggest impact on the TLS ecosystem, especially the long tail, is Let's Encrypt (letsencrypt.org), "a free, automated, and open Certificate Authority." While the cost-free issuance of certificates takes away any economic reason to not implement HTTPS, and the open design facilitates transparency and continuous monitoring, the automated fashion of Let's Encrypt highly improves the usability of certificate issuance. This corresponds to the setup phase (the first column in the image) of our identified TLS deployment process model. All setup phase steps are replaced with a single tool-supported step. The certificate issuance is completely automated with the ACME protocol, so no further manual input is needed and the major Web server software, e.g., Apache, is also integrated. Let's Encrypt changed the TLS ecosystem significantly, with now more than 100 million certificates issued.

Tool support for the second phase—the hardening phase—is also quickly improving. For choosing the appropriate ciphersuites and associated compatibility issues, Mozilla published their Mozilla SSL Configuration Generator [6]. With selected server software and a chosen compatibility mode, the tool generates a valid and sane configuration. This can easily be copied and pasted into the correct server configuration file. Complicated decisions are replaced with few more easily understandable options. The compatibility level has three profiles: old, intermediate, and modern. The tool shows the oldest compatible clients upon selection and also adds correct HSTS headers to the configuration. So it also unburdens the use of HSTS.

Further tool support exists for testing the deployed security configuration. For publicly reachable domains this enables iterative configuration with repeated security testing until a specific grade is reached. The most established testing tool is the Qualys SSL Server Test, the same tool we graded the participants' configurations. With Hardenize (hardenize.com) this concept is widened to DNS, email, and application security.

In our study, we addressed the ecological validity of our results by conducting additional expert interviews with experienced security consultants. One expert mentioned the Web server software Caddy (caddyserver.com). It comes with a secure TLS default configuration and automatically uses Let's Encrypt to retrieve certificates. Initially delivering HTTPS with your software is a good example of the paradigm of secure defaults. We have to shift this paradigm to other major Web server software.

This tool support, although not yet fully integrated, provides important assistance for administrators, but there is still a lot of work to do to shift the Internet to HTTPS.

Outlook

As mentioned, the Firefox telemetry data showed that more than 50% of Web pages are loaded over HTTPS. This is an enormous success, and the overall trend is definitely pointing in the right direction. We see a lot of effort to shift the ecosystem towards HTTPS. With TLSv1.3, not only the security but also the performance of TLS is increased, making HTTPS even more attractive. We see more and more hosting platforms switching to HTTPS for their customers, which is increasing HTTPS usage for the long tail at scale. New upcoming standards, like the ACMEv2 standard [7] are improving the automation of certificate issuance. And the future support of wildcard certificates with Let's Encrypt [8] will form the ecosystem even more.

Conclusion

Administrators of the long tail of the Internet should sometimes also be seen as users. Some configuration tasks still require a deeper understanding of security mechanisms or even underlying cryptographic methods. If we want these security mechanisms to work, we also have to support administrators in enabling them. With this in mind, we should all work on shifting the ecosystem, until even the long tail supports HTTPS so that we can finally move on to the Internet where HTTPS is the norm.

References

[1] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS Adoption on the Web," in *Proceedings of the 26th USENIX Security Symposium (Security '17)*, pp. 1323–1338: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-felt.pdf.

[2] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl, "'I Have No Idea What I'm Doing'—On the Usability of Deploying HTTPS," in *Proceedings of the 26th USENIX Security Symposium (Security '17)*, pp. 1339–1356: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-krombholz.pdf.

[3] Y. Zhu, "(mis)adventures in setting up HTTPS": https://www.youtube.com/watch?v=Q0VdlLG7t1w.

[4] Qualys SSL Labs, SSL Server Test: https://www.ssllabs.com/ssltest.

[5] I. Ristić, SSL Server Rating Guide: https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide.

[6] Mozilla SSL Configuration Generator: https://mozilla.github.io/server-side-tls/ssl-config-generator/.

[7] ACMEv2 API Endpoint Coming January 2018: https://letsencrypt.org/2017/06/14/acme-v2-api.html.

[8] Wildcard Certificates Coming January 2018: https://letsencrypt.org/2017/07/06/wildcard-certificates-coming-jan-2018.html.

Better Passwords through Science (and Neural Networks)

WILLIAM MELICHER, BLASE UR, SEAN M. SEGRETI, LUJO BAUER, NICOLAS CHRISTIN, AND LORRIE FAITH CRANOR

William Melicher is a PhD student in the College of Electrical and Computer Engineering at Carnegie Mellon University. He works on passwords, Web security, and online privacy. He received his undergraduate degree in computer engineering from the University of Virginia. billy@cmu.edu

Blase Ur is Neubauer Family Assistant Professor of Computer Science at the University of Chicago. His research focuses broadly on usable security and privacy, including authentication, privacy transparency, and tools for helping users make better security decisions. He received his PhD and MS from Carnegie Mellon University and his AB from Harvard University. blase@uchicago.edu

Sean Segreti is a Security Consultant, Developer, and Passwords Researcher at KoreLogic. Segreti maintains Carnegie Mellon University's Password Guessability Service, which is used by over 30 universities to estimate password strength. Segreti holds a master's degree in electrical and computer engineering (ECE) from Carnegie Mellon University, and a bachelor's degree in electrical engineering from the University of Maryland. ssegreti@cmu.edu

In this article, we discuss how we use neural networks to accurately measure password strength, and how we use this capability to build effective password meters. First, we show how neural networks can be used to guess passwords and how we leveraged this method to build a password guesser to better model guessing attacks. We report our measurements of the effectiveness of neural networks at guessing passwords, demonstrating that they outperform other popular methods of modeling adversarial password guessing. We then show how we developed a password guesser that can be compressed so that it is practical for client-side use inside a Web page [1]. Finally, we describe how we designed and built a password meter, based on neural networks, that gives more accurate and helpful guidance to users for creating passwords that are resistant to guessing attacks [2].

Passwords are the most common authentication mechanism in use today. We all use passwords every day and will likely continue to do so for the foreseeable future. Unfortunately, human-chosen passwords often follow predictable patterns. For example: exclamation points are at the end; capital letters are at the beginning of passwords; dictionary words, well-known phrases, keyboard patterns, and names of people and places are all common. Such predictable patterns allow attackers to break into accounts by guessing passwords.

Guessing attacks can take the form of online attacks in which attackers make guesses while trying to log in to a live system. Online attacks are sometimes defended against by limiting the rate at which attackers may make guesses against the system. In contrast, in offline guessing attacks, attackers can make large numbers of guesses without limits. This commonly happens when a database of hashed passwords is stolen, an event that occurs with disappointing regularity. Attackers guess candidate passwords and compare them against hashed passwords in the database, limited only by the amount of computer resources they have. The widespread incidence of password reuse makes such attacks more dangerous because attackers who crack a user's password that was leaked from a stolen database may use that cracked password—or common variations of the password—to guess the credentials for that user's other accounts. A common and effective defense against both online and offline guessing attacks is to urge users to create less predictable passwords that are more resistant to guessing.

To understand how to guide users to make less guessable passwords, our research group has studied methods for modeling how attackers guess passwords. Previous approaches for modeling password-guessing attacks include statistical approaches, and tools used in adversarial password cracking. Statistical methods, such as Markov models and probabilistic context-free grammars, work by deriving statistical properties from lists of training passwords. Adversarial password cracking tools, such as John the Ripper and Hashcat, are typically used in practice for their ability to crack hashed passwords quickly; often they are configured by experts to craft special password cracking rules for specific password sets. Prior work from our group has studied these approaches and shown how the combination

Lujo Bauer is an Associate Professor in the Electrical and Computer Engineering Department and in the Institute for Software Research at Carnegie Mellon University. His research interests span many areas of computer security and privacy, and include building usable access-control systems with sound theoretical underpinnings, developing languages and systems for run-time enforcement of security policies on programs, and generally narrowing the gap between a formal model and a practical, usable system. His recent work focuses on developing tools and guidance to help users stay safer online and in examining how advances in machine learning can lead to a more secure future. lbauer@cmu.edu

Nicolas Christin is an Associate Research Professor at Carnegie Mellon University, jointly appointed in the School of Computer Science and in Engineering and Public Policy. He holds MS and PhD degrees in computer science from the University of Virginia. His research interests are in computer and information systems security; most of his work is at the boundary of systems and policy research. He has most recently focused on security analytics, online crime modeling, and economics and human aspects of computer security. nicolasc@cmu.edu

Lorrie Faith Cranor is a Professor of Computer Science and of Engineering and Public Policy at Carnegie Mellon University where she is director of the CyLab Usable Privacy and Security Laboratory (CUPS). She is Associate Department Head of the Engineering and Public Policy Department and Co-Director of the MSIT-Privacy Engineering masters program. In 2016 she served as Chief Technologist at the US Federal Trade Commission. She is also a co-founder of Wombat Security Technologies, Inc., a security-awareness training company. She is a fellow of the ACM and IEEE and a member of the ACM CHI Academy. lorrie@cmu.edu

of multiple automated approaches approximates the ability of professional human experts to guess passwords [3]. However, modeling a guessing attack in which attackers can make large numbers of guesses often requires servers with tens of CPU cores and with gigabytes of disk space for storing models of password guessing. Such models are not practical for giving real-time feedback to users during password creation; users can't download gigabytes of data or wait days or weeks to get feedback for creating a password.

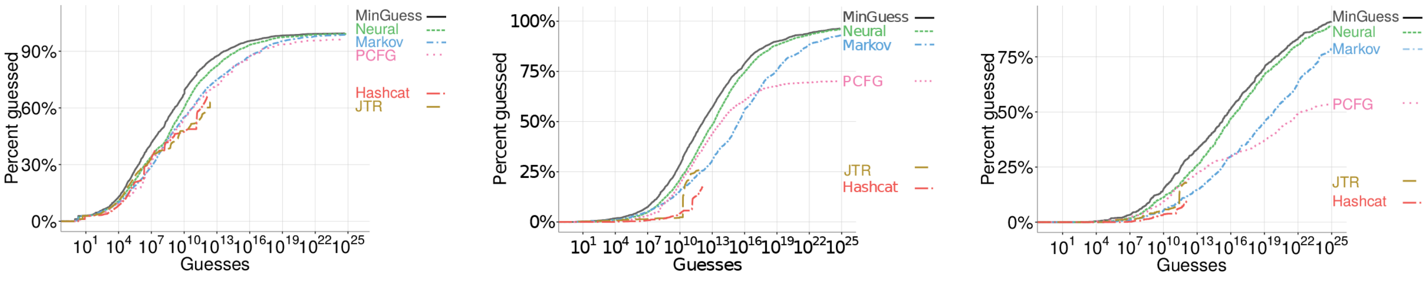
Due to the challenges of accurately modeling password attacks, most password meters are unable to provide data-driven, principled feedback to users during password creation. Meters will typically calculate some combination of a variety of heuristics—such as the number of special characters used or the length of the password—which often has little correlation to the resistance of passwords to guessing attacks [4]. When faced with such meters, users often make predictable modifications in order to satisfy the meter's strength estimate, such as adding an exclamation point to the end of their password. However, because attackers are also aware of the predictable patterns in password construction, such modifications do little to improve the password's resistance to guessing. In addition, meters are often incapable of providing positive advice or giving users suggestions about how to make passwords better, instead rating a password as simply "weak" or "fair."

Designing a Neural Network Guesser

Neural networks are a machine-learning technique that is particularly adept at fuzzy classification problems and problems dealing with computer processing of natural language. The intuition for our approach was that, because the task of guessing passwords in an adversarial attack is conceptually related to generating natural language, neural networks would be well suited to our goal of modeling guessing attacks. Recently, the machine-learning community has showed how to use neural networks to generate text, which our approach leverages [5]. Generating a password with a neural network involves repeatedly predicting the next character of a password to build up the password one character at a time. This process can be extended to generate large numbers of probable passwords. During training, the neural network is taught to predict the next character when given a real password fragment. The neural network can then learn to recognize high-level patterns that often arise in password construction, such as keyboard patterns or exclamation points at the end of a password.

We tried many different variations and tunings for training our neural network guesser. When training neural networks, there is a large design space of different parameters and design decisions to explore for better performance. We experimented with a wide range of different parameters including: the number of parameters in the model; the method of representing password characters; different recurrent neural-network architectures; using different types of training data; and using a technique called transference learning, which specializes neural network predictions for different situations. At the end of these experiments, we had a neural-network training methodology that we found was most accurate for our application of guessing passwords. Additionally, we used a technique of modeling password guessing to arbitrarily high numbers of guesses by employing Monte Carlo methods [6], allowing us to accurately model password guessability against nation-states or other extremely powerful adversaries who have huge resources for cracking passwords.

When designing our neural-network guessing method, we tested it against the best tunings of other methods for guessing passwords. In addition, during development of our neural-network guesser, we comprehensively tested various different versions of the neural-network guesser against each other to find the best method. We measured the performance of our guessing approaches both on real passwords collected in recent password leaks and on passwords we have collected in our research studies, allowing us to compare the performance of guessing methods in a wide variety of password policies and situations. To train our guessing



(a) Guessing passwords that must be more than eight characters

(b) Guessing passwords that are required to be more than eight characters long and have a mix of character classes

(c) Guessing passwords that are required to be more than 12 characters long and have a mix of character classes

Figure 1: Comparison of the ability of different password methods to guess passwords. The x-axis of each graph shows the number of guesses made in log scale. The y-axis shows the percent of passwords guessed. Higher lines on the graph represent more accurate guessing. "Neural" shows the performance of our neural-network approach; "Markov" the Markov model approach; "PCFG" probabilistic context-free grammars; "JTR" John the Ripper; "Hashcat" shows the performance of Hashcat; and "MinGuess" shows a combination of all approaches, where a password receives the minimum guess number from all approaches. Each graph shows passwords created under a different policy—requiring a different minimum length and different mix of character classes (uppercase and lowercase characters, digits, and symbols).

methods in our experiments, we required large numbers of real passwords, which we obtained from leaked password lists. In total, our data set of passwords contained over 100 million passwords from more than 20 password leaks. This huge amount of data on real-world passwords allows machine-learning techniques to infer deep insights into password construction and to have the predictive power to model common password patterns.

We found that the neural networks guessed passwords more accurately than any other individual method. However, while our best-performing neural networks often performed close to an optimal guessing strategy, the combination of all methods including neural networks (MinGuess in Figure 1) performed better than just neural networks alone, showing that a combination of many models is still better than any individual method. Nonetheless, if one is limited to only one method for estimating password strength, neural networks are the most accurate. Figure 1 shows a selection of some of our results on guessing accuracy for different conditions; the neural network approach guesses a larger proportion of passwords over the same number of guesses than other methods. This finding holds to various degrees across all of our test sets, although we find that neural networks are particularly accurate when guessing passwords made under the more exotic, stronger password policies, which are becoming increasingly common as password guessing abilities increase.

Designing a Client-Side Strength Estimator

Besides increasing the accuracy of existing password strength models, we also strove to develop more practical models. Previous methods for modeling adversarial password cracking require large amounts of disk space or bandwidth—hundreds of megabytes or gigabytes—and take hours or days to calculate measures of password strength. In contrast, to give real-time feedback to users during password creation, models must be smaller to download and give quick results. For this application, we wanted a model that was less than one megabyte to download, which is roughly half the size of an average Web page. Additionally, in the context of real-time feedback, a model must calculate a measurement of password strength within a fraction of a second—ideally below the threshold of human recognition, which is roughly 100 ms. In addition to these properties, the measurement should be accurate, and the model should run inside of a Web browser, which means that JavaScript is the most viable execution platform.

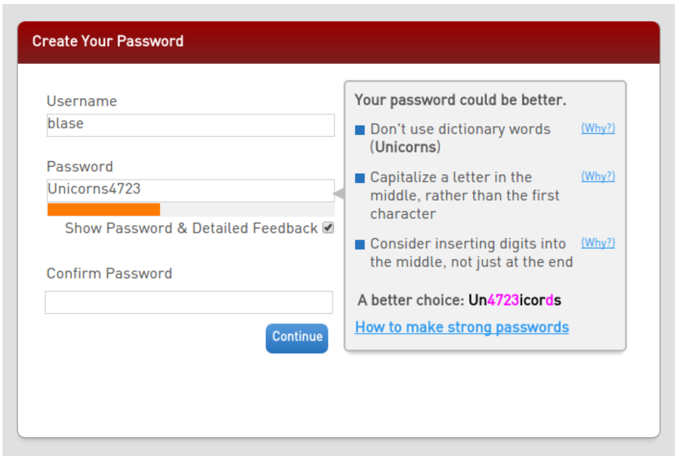
Given the challenges of implementing accurate password-strength measurement on resource-constrained clients, it might be tempting to use a system architecture where the password model is stored on a server and only measurement results are communicated to the client. However, in many situations, the user's password should never be sent to the server for security reasons, for example, in the case of device encryption software, keys that protect cryptographic credentials, or the master password for a password manager. Even in cases where the user's password is eventually sent to an external server, using a remote password-strength measurement mechanism may allow powerful side channels based on keyboard timing, message size, and caching [7]. For these reasons, we preferred architectures where password modeling and strength estimation are done entirely on the client side. This design decision has the added benefit of being easier for Web administrators to deploy.

To summarize our technical approach to meeting these goals: we started by training a neural network with fewer parameters—the features of the model that define how to predict the next character. Using this less complex model made the network smaller, but did not sacrifice much accuracy compared to our best-performing network. Then we reduced the precision of the already shrunken neural network's parameters, again trading off space for some accuracy. Finally, we used standard lossless compression methods to further shrink the size of the model, eventually reaching a model size of 850 KB. To make our network produce low-latency results, we pre-computed an approximate mapping for estimating the strength of the password, which was sent to the client along with the network. In addition, we cached specific intermediate computations, so that the common case, in which a character is added to the end of the password, is quicker because the strength estimator only needs to update its previous computation. We were able to get the average response time to be 17 ms for this common case. Some of our optimizations sacrificed accuracy for the sake of quicker results or a smaller model; we empirically measured the impact that such optimizations introduced and found the error rate to be small enough to be acceptable for our purposes. In addition, we tuned the network so that it was much more likely that we would make safe errors—underestimating a password's strength—than unsafe errors.

We compared the accuracy of our client-side strength estimation based on neural networks to existing password meters: "zxcvbn" and Yahoo's password meter. zxcvbn, in particular, measures password strength using a number of highly tuned heuristics for password strength. We found our method of measuring password strength to be more accurate—correlating more highly with password strength measured by simulating a guessing attack—than either meter, having between 39% and 83% fewer unsafe errors, depending on the meter and the password policy. At the same time, our strength measurement also had fewer safe errors. In addition, our more principled method of simulating adversarial guessing entirely on the client-side has the benefit that it can be easily reconfigured—by re-training the neural network—for new password policies or new situations. We know that certain password sets often have special patterns that are unique to that set: for example, passwords for a sports Web site may contain more sports terminology than other password sets. Our method would be able to be easily retargeted to learn such patterns.

Design of a Password Meter

While the development of an accurate client-side strength-estimation tool is necessary for a password meter, it is not sufficient. There is a gap between a practical measurement of strength and providing effective real-time feedback about how to make a better password. We wanted to bridge this gap. Our main goal was to give human-understandable feedback about password creation; our neural-network strength estimation by itself can tell the user that a password is weak or strong, but it cannot say how to improve the password to be more resistant to guessing. To accomplish this, we aimed to give two types of suggestions: first, we wanted to be able to provide concrete suggestions for specific passwords that are stronger; second, we wanted to provide users with high-level guidance specific to their exact



Create Your Password

Username
blase

Password
Unicorns4723

Show Password & Detailed Feedback ☑

Confirm Password

Continue

Your password could be better.

- Don't use dictionary words (Unicorns) (Why?)
- Capitalize a letter in the middle, rather than the first character (Why?)
- Consider inserting digits into the middle, not just at the end (Why?)

A better choice: Un4723icords

How to make strong passwords

Figure 2: Screenshot of our password meter's interface. The bar shows the strength estimate of the user's password. The popup dialog shows specific password feedback based on the user's password.

situation—for example, notifying users that using capital letters at the beginning of the password is a common pattern and does not meaningfully improve the strength of their password.

We developed a password meter that achieves these goals. Our meter combines the accuracy of our neural-network strength measurement with a series of data-driven heuristics that provide human-understandable feedback about the user's password. Figure 2 shows an example of our meter in action. Our meter uses the neural network to control the bar that shows how strong the user's password is, while data-driven heuristics additionally give the user specific feedback about how to improve their password. The meter can also provide a concrete suggestion for how to change the password so that it will be stronger. It does so by creating several candidate suggestions that are similar to the user's chosen password and then using the neural network to gauge their strength. Only those candidate passwords that are judged stronger by the network are shown to the user.

We tested whether the meter helps users to create stronger passwords. We recruited participants to create a password for a hypothetical high-value online account in a variety of different conditions—some participants used our meter during password creation, some used modified versions of our meter, and some did not have the benefit of any meter. Similar methodology has been used in prior work by our group for measuring the impact of a variety of different conditions on the security and usability of human-chosen passwords [8, 9].

We found that participants who used the meter created passwords that were 44% more resistant to guessing attacks than those who did not. Interestingly, we also found that participants who saw the human-readable suggestions produced even stronger passwords than those who only saw the measurement of strength. This implies that not only does providing real-time strength estimates help users make stronger passwords, but also that providing actionable suggestions about what users should do provides additional benefit.

Conclusion

We showed how neural networks can be used to guess passwords and that they can do so more accurately than other methods for adversarial password guessing. We also showed how leveraging neural networks can lead to more practical estimations of password strength on resource-constrained client machines in real time. Finally, we built and tested a password meter, based on neural networks, that gives human-understandable feedback and guides users to make better passwords. We have released our meter as open source software (at https://github.com/cupslab/neural_network_cracking and https://github.com/cupslab/password_meter) and invite people to use it.

Acknowledgments
We would like to thank Mahmood Sharif for participating in discussions about neural networks and Dan Wheeler for his feedback. This work was supported in part by gifts from the PNC Center for Financial Services Innovation, Microsoft Research, John & Claire Bertucci, and a gift from NATO through Carnegie Mellon CyLab.

References

[1] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, L. F. Cranor, "Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks," in *Proceedings of 25th USENIX Security Symposium,* 2016: http://bit.ly/2fB18Jd.

[2] B. Ur, F. Alfieri, M. Aung, L. Bauer, N. Christin, J. Colnago, L. F. Cranor, H. Dixon, P. E. Naeini, H. Habib, N. Johnson, W. Melicher, "Design and Evaluation of a Data-Driven Password Meter," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems,* ACM, 2017: https://doi.org/10.1145/3025453.3026050.

[3] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, R. Shay, "Measuring Real-World Accuracies and Biases in Modeling Password Guessability," in *Proceedings of the 24th USENIX Security Symposium,* 2015: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-ur.pdf.

[4] X. de Carné de Carnavalet and M. Mannan. "From Very Weak to Very Strong: Analyzing Password-Strength Meters," in Proceedings of the 18th Network and Distributed System Security Symposium, 2014: https://www.ndss-symposium.org/ndss2014/programme/very-weak-very-strong-analyzing-password-strength-meters/.

[5] I. Sutskever, J. Martens, and G. E. Hinton. "Generating Text with Recurrent Neural Networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11),* http://www.icml-2011.org/papers/524_icmlpaper.pdf.

[6] M. Dell'Amico and M. Filippone, "Monte Carlo Strength Evaluation: Fast and Reliable Password Checking," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security,* 2015: https://doi.org/10.1145/2810103.2813631.

[7] D. X. Song, D. Wagner, and X. Tian, "Timing Analysis of Keystrokes and Timing Attacks on SSH," in *Proceedings of the 10th USENIX Security Symposium,* 2001: https://www.usenix.org/legacy/events/sec01/full_papers/song/song.pdf.

[8] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman, "Of Passwords and People: Measuring the Effect of Password-Composition Policies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* ACM, 2011: https://doi.org/10.1145/1978942.1979321.

[9] R. Shay, S. Komanduri, A. L. Durity, P. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, "Can Long Passwords Be Secure and Usable?" in *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems,* 2014: https://doi.org/10.1145/2556288.2557377.

The Road to Scalable Blockchain Designs

SHEHAR BANO, MUSTAFA AL-BASSAM, AND GEORGE DANEZIS

Shehar Bano is a Postdoctoral Researcher at University College London. Her research interests center on networked systems, particularly in the context of security and measurement. She received her PhD from the University of Cambridge in 2017 where she was an Honorary Cambridge Trust Scholar, and was awarded the Mary Bradburn Scholarship for her research work. s.bano@ucl.ac.uk

Mustafa Al-Bassam is a PhD student at University College London, working on scalable distributed ledger technology and peer-to-peer systems. He received a BSc in computer science from King's College London, where his final-year project focused on public-key infrastructure implemented with smart contracts. mustafa.al-bassam.16@ucl.ac.uk

George Danezis is a Professor of Security and Privacy Engineering at University College London and a Turing Faculty Fellow, where he heads the Information Security Research group. He researches privacy-enhancing technologies, decentralization, and infrastructure security and privacy. In the past he worked at Microsoft Research, KU Leuven, and the University of Cambridge, where he also studied. g.danezis@ucl.ac.uk

Bitcoin has become centralized and slow due to the inherent limitations of its blockchain. A number of alternative blockchain designs have been proposed to address these issues. *Off-chain* solutions allow for small and frequent transactions to take place over low-tier blockchain instances, parallel to and backed by the main blockchain. *On-chain* solutions directly modify the blockchain design to support high performance. We focus on the latter and summarize and discuss recent approaches to on-chain scaling of blockchains.

Despite being founded on the ideals of openness and freedom of information, the Internet has become increasingly centralized, enabling a small number of big players to control who can access information. A number of proposals have emerged to counterbalance this trend such that information storage and processing is not concentrated in any single entity. Of these, blockchains are particularly promising, capturing the attention of popular media, research, and policy communities alike.

A blockchain is an immutable and decentralized database that facilitates transparent and auditable management of data. It first gained traction as the underlying technology of Bitcoin [8] proposed in 2009, but it has since independently evolved thanks to its properties of resiliency, integrity, and transparency. Yet Bitcoin suffers from scalability issues that impede its wider adoption. Over the past year, major divisions have emerged in the Bitcoin community over how Bitcoin should scale, as blocks of transactions have reached capacity, resulting in transaction fees skyrocketing. At the core of the debate is a simple tradeoff between scalability and centralization: the bigger the blockchain, the fewer devices will have the capacity to store and audit the full blockchain, leading to the network becoming more centralized.

Some argue for using the Bitcoin blockchain as a settlement network for large transactions only—with smaller transactions being handled by payment hubs off the blockchain (off-chain scaling), while others argue for increasing the capacity of the blockchain itself for all types of transactions (on-chain scaling). As a result of the fallout, proponents of on-chain scaling recently forked the Bitcoin blockchain to create their own network called Bitcoin Cash. In this article, we provide an overview of key themes and options for on-chain scaling of blockchains.

Functional Components of a Blockchain

A blockchain serves as a decentralized database—or a distributed *ledger*—representing a consensus of synchronized, distributed, and replicated data (called *blocks*, representing sets of *transactions*). It is internally implemented as a linked list in which pointers to previous blocks have been replaced with cryptographic hash pointers (Figure 1). A pointer is simply the hash of some information (e.g., the previous block in this case) and serves to identify the information, as well as to verify its integrity. Each block in the blockchain contains a hash of the previous block and information specific to the current block. The resulting hash chain ensures each block implicitly verifies integrity of the entire blockchain before it. Thus,

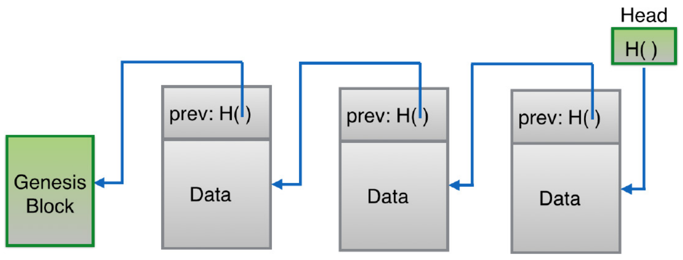


Figure 1: A blockchain is implemented as a linked list of hash pointers.

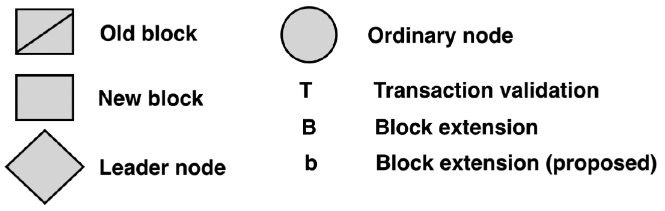


Figure 2: Legend used in the figures

a blockchain acts as a tamper-evident log where data can be appended to the end of the log, and tampering with previous data in the log is detectable. A blockchain has two key functional components: *transaction validation* and *extending the blockchain*, which we discuss individually.

Transaction Validation

A *transaction* specifies some transformation on the state of the ledger. These transactions, subject to passing validity and verification checks, are included in a candidate block (a set of transactions) to be appended to the blockchain. As a concrete example, we describe a typical (simplified) Bitcoin transaction involving transfer of money from payer(s) to payee(s). The payers and payees are identified by their public keys, and the payer digitally signs the transaction, involving value they control encoded in previous blocks of the blockchain. Nodes in the Bitcoin network must perform a set of checks before accepting a transaction as valid according to the rules of the network. First, they must check that the transaction is well-formed. Second, they must verify that the payer is authorized to conduct this transaction by checking that its digital signature corresponds to the public key of the payer. Third, the nodes must verify that the sum of outputs is lower than the sum of inputs—payers cannot pay out more than they own, but a transaction fee can be included in the payment. Finally, nodes must ensure that none of the inputs is being double-spent. This can be verified by traversing back in the blockchain to when the input value was created, and then traversing forward all the way to the current transaction—ensuring along the way that the input has not been previously spent.

Extending the Blockchain

In reality, transaction outputs (e.g., *X bitcoins*) have no physical existence: the fact that Bob owns a transaction output corresponds to the fact that a majority of the nodes believe this to be the case. Agreement between nodes on how to extend the blockchain is reached through a collaborative process called *consensus*.

Consensus. The problem of consensus in the presence of faulty or malicious nodes has seen extensive study in the distributed systems community, long before it was revisited in the context of blockchains. In a network with *n* honest nodes that each receive input values and share them with rest of the network, the consensus protocol enables agreement between all *n* honest nodes on the set of input values generated by honest nodes. In the Bitcoin context, where nodes broadcast transactions as part of a peer-to-peer (p2p) network, nodes need to reach consensus on exactly which transactions took place and in what order—that is, the nodes must agree on the state of the blockchain.

Forks. Consensus is challenging because nodes might have different views of the blockchain (*forks*) due to latency in propagation of transactions over the p2p network, nodes randomly failing, and malicious nodes trying to suppress valid transactions and push invalid transactions to the blockchain. Forks defy consensus, so a mechanism is needed to resolve conflicts and get a majority of the nodes to agree on the state of the blockchain.

Leader. Consensus protocols typically rely on a *leader*. The leader is responsible for coordinating with other nodes to reach consensus, and for appending a final, committed value to the blockchain. The leader is usually effective only for a period of time called an *epoch*, after which or upon a fault, a new leader is elected. A crucial property of a leader is that it should behave honestly. This is important because even though a blockchain is tamper-evident by design, appending bad blocks to the blockchain will likely result in forks leading to wasted system resources in getting nodes to re-converge to a previous valid blockchain view. Honest leader behavior is usually enforced via incentivization and auditability.

Figure 2 shows visual representations of some of these concepts, which are later used to explain various design themes.

Bitcoin and Its Scalability Issues

The advent of Bitcoin in 2009 sparked interest in blockchains, the technology that lies at its foundation. Being the predecessor of the myriad blockchain variations that subsequently emerged, it is useful to understand the blockchain scalability problem in the context of Bitcoin.

The Bitcoin Blockchain. Bitcoin is a p2p network where any node can join and become part of the network. If a node receives

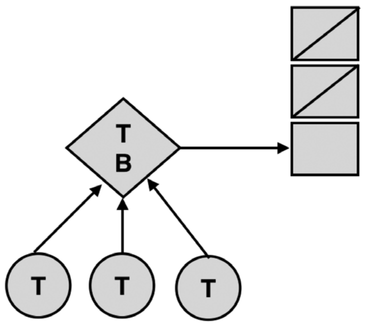


Figure 3: The Bitcoin blockchain model

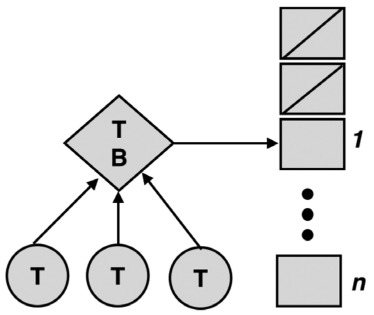


Figure 4: Multiple blocks per leader

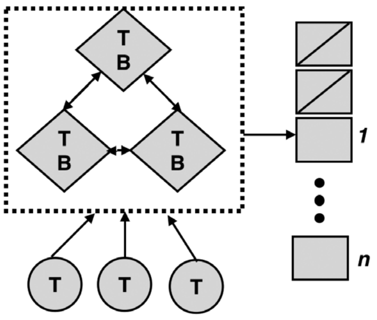


Figure 5: Collective leaders

a new block, it broadcasts it to rest of the network (Figure 3). While all nodes listen to and broadcast blocks, only leader nodes can append information to the blockchain. To stop dishonest leaders from bringing the system to a stall—for example, by creating frequent forks—the leader for each epoch is chosen randomly via proof-of-work. This involves solving a hash puzzle—also called *mining*, which is why Bitcoin leaders are referred to as *miners*. If a miner gets lucky by finding a solution to the hash puzzle, it proposes the next block to append to the blockchain. To incentivize miners to solve hash puzzles and propose next blocks, successful miners are rewarded by allowing them to pay some amount to themselves (a *block reward*, which diminishes over time) or by keeping some part of the transaction output amount as the *transaction fee*.

Blockchain Scalability. Two metrics are directly related to blockchain scalability: transaction throughput (the maximum rate at which the blockchain can process transactions) and latency (time to confirm that a transaction has been included in the blockchain). While previous work has identified additional metrics [4], throughput and latency are bottleneck issues and more challenging to address from a research perspective. Bitcoin's transaction throughput is a function of its block size and inter-block interval. With its current block size of 1 MB and 10 minute inter-block interval, the maximum throughput is capped at about seven transactions per second; and a client that creates a transaction has to wait for at least 10 minutes on average to be sure that the transaction is included in the blockchain. In contrast, mainstream payment-processing companies like Visa confirm transactions within a few seconds and have a high throughput of up to 24,000 transactions per second [9].

Current research is focused on developing solutions to significantly improve blockchain performance while retaining its decentralized nature. Reparametrization of Bitcoin's block size and inter-block interval can improve performance to a limited extent—estimated by a recent study [4] at 27 transactions per second and 12 seconds, respectively. However, significant improvement in performance requires fundamental redesign of the blockchain paradigm.

Redesigning Blockchains for Scalability

We now take a look at key design schemes that have been developed to improve blockchain scalability. Our scope is restricted to approaches targeting the blockchain core design (on-chain solutions) rather than techniques that delegate trust to parallel off-path blockchain instances such as sidechains [1] (off-chain solutions). The list of themes and example systems is not meant to be comprehensive but, rather, indicative of some major ways in which this subject has been approached, and to provide a high-level roadmap for future research efforts. Figure 2 shows the basic building blocks of the design themes that we discuss in the following sections.

Multiple Blocks per Leader

Bitcoin-NG [5] shares Bitcoin's trust model but decouples leader election (performed randomly and infrequently via proof-of-work) from transaction serialization (Figure 4). However, unlike Bitcoin where the leader can only propose one block to append to the blockchain, Bitcoin-NG divides time into epochs, and a leader can unilaterally append multiple transactions to the blockchain for the duration of its epoch, which ends when a new leader is elected. There are two kind of blocks in Bitcoin-NG: keyblocks and microblocks. Keyblocks contain a solution to the puzzle and are used for leader election. Keyblocks contain a public key that is used to sign subsequent microblocks generated by the leader. Every block contains a reference to the previous microblock and keyblock. A fee is distributed between the current leader (40%) and the next leader (60%).

Similar to Bitcoin, forks are resolved by extending the longest branch aggregated over all keyblocks. Note that microblocks do not contribute to the length of a branch since these do not contain proof-of-work. To penalize a leader that creates forks in the generated microblocks, a subsequent leader can insert a special poison transaction after its keyblock that contains the header of the first block in the pruned branch as a proof-of-fraud. This invalidates the malicious leader's reward, a fraction of which is paid to the reporting leader. Forks can also occur when a new leader has been elected but the previous leader has not yet heard about it and continues to generate microblocks. However, such forks are resolved as soon as the announcement of the new leader election reaches all the nodes.

Collective Leaders

This scheme employs multiple leaders to collectively and quickly decide if a block should be added to the blockchain (Figure 5). ByzCoin [6] replaces Bitcoin's probabilistic transaction consistency guarantees with strong consistency by extending Bitcoin-NG (see preceding section) to achieve high transaction throughput. This has the advantage that a transaction submitted by a client will be added to the blockchain, and the blockchain remains fork-free since all leaders instantly agree on block validity. ByzCoin modifies how Bitcoin-NG generates keyblocks: a group of leaders, rather than a single leader, generates a keyblock followed by microblocks. The leader group is dynamically formed by a window of recent miners. Each miner has voting power proportional to the number of mining blocks it has in the current window, which is its hash power. When a new miner solves the puzzle, it becomes a member of the current leader group, which moves one step forward, ejecting the oldest miner. ByzCoin uses the same incentive model as Bitcoin, but the remuneration is shared between members of the leader group in proportion to their shares.

The leader group is organized into a communication tree where the most recent miner (the leader) is at the root. The leader runs a modified version of the Practical Byzantine Fault Tolerance (PBFT) protocol [3] with linear messaging complexity to generate a collective signature that proves that at least two-thirds of the consensus group members witnessed and attested the microblock. A node in the network can verify in $O(1)$ that a microblock has been validated by the consensus group. This design addresses a limitation of Bitcoin-NG where a malicious leader can create microblock forks: in ByzCoin this would require a two-thirds majority of leader group members to be malicious. Moreover, Bitcoin-NG suffers from a race condition where an old leader who has not yet heard about the new leader may continue to incorrectly mine on top of older microblocks. In ByzCoin, leader group members ensure that a new leader builds on top of the most recent microblock.

Parallel Blockchain Extension

As shown in Figure 6, in this approach multiple leaders extend in parallel different parts of the blockchain (e.g., represented as a graph of transactions). Bitcoin has a linear process of extending the blockchain: miners try to solve the puzzle, and the one that finds a solution appends the next block. The framework proposed by Boyen, Carr, and Haines [2] parallelizes this process by forgoing the concepts of "blocks" and "chain" in favor of a graph of cross-verifying transactions. Each transaction validates two previous transactions (its parents) and contains some payload (e.g., cryptocurrency) and proof-of-work.

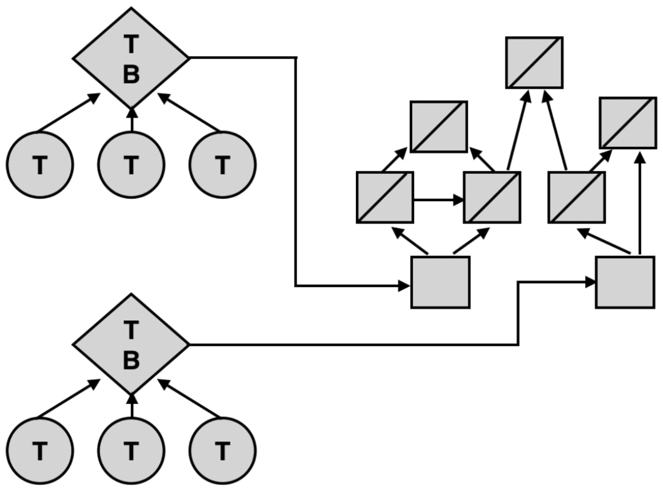


Figure 6: Parallel blockchain extension

A transaction can be potentially validated by multiple children nodes. Additionally, each transaction also carries a reward to be collected by the transaction that validates it. The value of the reward decreases as more nodes directly or indirectly validate it, so new nodes have more incentive to validate recent transactions. The system has been shown to converge, meaning that at some point there is a transaction that connects to (and thus implicitly verifies) all transactions before it. As a result of this graph structure, miners can extend different branches of the transactions graph in parallel. Normal (non-miner) nodes in the system verify transactions as they receive them. In addition to standard checks on the correctness of proof-of-work and structural validity of the transaction and its parents, the node also checks that the transaction is not a double-spend by accepting as valid the well-formed transaction that has the largest amount of work attached to it.

Sharding Transactions

Elastico [7] partitions nodes into groups called committees, and each committee manages a subset (shard) of transactions. In Figure 7, the top shard handles the first 10 transactions, while the bottom shard handles the next 10. Within a committee, nodes run a Byzantine consensus protocol (e.g., PBFT) to agree on a block of transactions. If the block has been signed by enough nodes, the committee sends it to a final committee. The final committee collates sets of transactions received from committees into a final block, runs a Byzantine consensus protocol between its members to get agreement on extending the blockchain, and broadcasts the appended block to other committees.

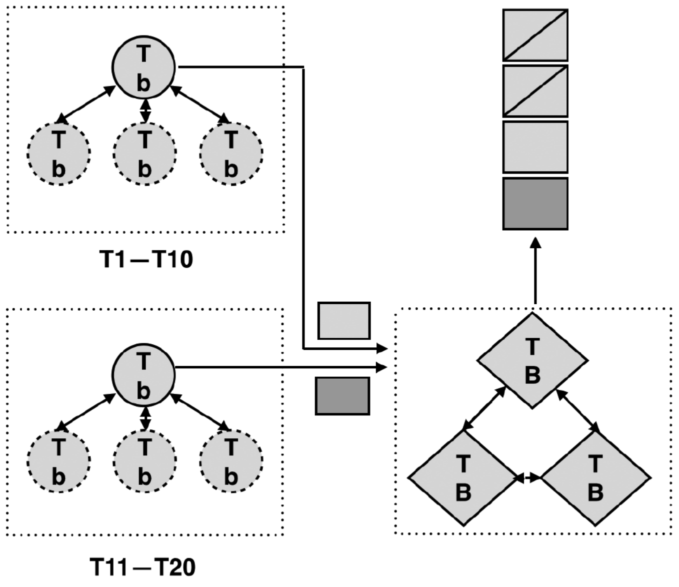


Figure 7: Sharding transactions

The system operates in epochs: the assignment of nodes to committees is valid only for duration of the epoch. At the end of the epoch, the nodes solve a puzzle seeded by a random string generated by the current final committee, and sends the solution to its next final committee. As a result, in each epoch a node is paired with different nodes in a committee and manages a different set of transactions. The number of committees scales linearly to the amount of computational power available in the system, but the number of nodes within a committee is fixed. Consequently, as more nodes join the network, the transaction throughput increases without adding to latency, since messages needed for consensus are decoupled from computation and broadcast of the final block to be added to the blockchain.

Conclusion

We framed the blockchain scalability problem and presented an overview of key approaches for on-chain scalability of blockchains. This revealed design patterns that can be used to compose scalable blockchains. Indeed, some patterns have already been used in this way: ByzCoin builds *collective leadership* on top of Bitcoin-NG's *multiple-blocks-per-leader* design as discussed. *Collective leadership* is a useful primitive to enforce honest behavior (and to avoid forks) by spreading out accountability and stakes across multiple leaders. *Sharding* speeds up transaction throughput by partly delegating consensus to smaller groups where classical BFT protocols can be effectively run, and making a leader group (that potentially also runs a BFT consensus protocol among leaders) responsible for extending the blockchain. It might be possible to replace consensus in the leader group with *collective leadership,* which has lower messaging complexity than the original PBFT protocol and a higher degree of trust. The idea of *parallel blockchain extension* can be combined with *sharding* such that the blockchain exists as partially connected trees on separate shards. Blocks that are part of separate trees are connected only when there is a transaction that consumes blocks managed by different shards.

Mining centralization is a well-known problem in Bitcoin: the biggest miners have built up a large advantage in how the blockchain grows. Systems like Bitcoin-NG and ByzCoin that inherit Bitcoin's mining-based consensus suffer from the same problem of centralization and favoring the biggest miners. Broadly, there has been a shift from Bitcoin's slow mining-based leader election to novel compositions or variations of classical consensus protocols. The latter cannot be directly employed in blockchains as these were originally written for a LAN setting, and their throughput decreases with the number of nodes. It will be interesting to see what new designs emerge and how existing consensus protocols are repurposed to operate in a decentralized WAN setting and in various threat models. This research direction revitalizes the field of Byzantine consensus and has the potential to make it relevant to widely deployed peer-to-peer systems.

Acknowledgments

The authors are supported in part by EPSRC Grant EP/M013286/1 and the EU H2020 DECODE project under grant agreement number 732546, as well as The Alan Turing Institute.

References

[1] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling Blockchain Innovations with Pegged Sidechains," Blockstream.com, 2014: https://www.blockstream.com/sidechains.pdf.

[2] X. Boyen, C. Carr, and T. Haines, "Blockchain-Free Cryptocurrencies: A Rational Framework for Truly Decentralised Fast Transactions," Cryptology ePrint Archive, Report 2016/871, 2016: https://eprint.iacr.org/2016/871.

[3] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, USENIX Association, 1999, pp. 173–186: http://bit.ly/2fjh6dN.

[4] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. G. Sirer, D. Song, R. Wattenhofer, "On Scaling Decentralized Blockchains," 3rd Workshop on Bitcoin and Blockchain Research, 2016: http://bit.ly/2xfz5Jl.

[5] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-NG: A Scalable Blockchain Protocol," in *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation (NSDI '16)*, pp. 45–59: http://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-eyal.pdf.

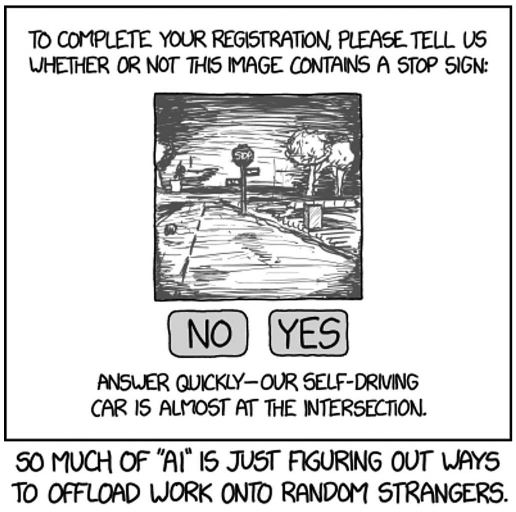
[6] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security '16)*, pp. 279–296: http://bit.ly/2wziEbl.

[7] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A Secure Sharding Protocol for Open Blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, pp. 17–30: https://www.comp.nus.edu.sg/~loiluu/papers/elastico.pdf.

[8] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," December 2008: https://bitcoin.org/bitcoin.pdf.

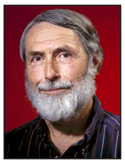
[9] https://usa.visa.com/run-your-business/small-business-tools/retail.html).

XKCD xkcd.com

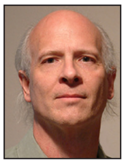

TO COMPLETE YOUR REGISTRATION, PLEASE TELL US WHETHER OR NOT THIS IMAGE CONTAINS A STOP SIGN:
STOP
NO
YES
ANSWER QUICKLY—OUR SELF-DRIVING CAR IS ALMOST AT THE INTERSECTION.
SO MUCH OF "AI" IS JUST FIGURING OUT WAYS TO OFFLOAD WORK ONTO RANDOM STRANGERS.

An Interview with Peter G. Neumann

RIK FARROW



Peter G. Neumann is Senior Principal Scientist in the SRI International Computer Science Department, where he has been for 45 years. His research has been concerned with computer systems and networks, trustworthiness/dependability, high assurance, security, reliability, survivability, safety, and many risk-related issues such as election-system integrity, cryptographic applications and policies, health care, social implications, and human needs—especially those including privacy. Currently, he is Principal Investigator of the joint SRI/Cambridge project relating to the CHERI system. He was at Bell Labs in Murray Hill, New Jersey, where he was heavily involved in Multics development. He has AM, SM, and PhD degrees from Harvard, and a Doctor rerum naturalium from Darmstadt. He moderates the ACM Risks Digest forum (http://www.risks.org) and has been responsible for 242 "Inside Risks" columns in the *Communications of the ACM*. He chairs the ACM Committee on Computers and Public Policy. His 1995 book, *Computer-Related Risks,* is still timely. He received the National Computer System Security Award in 2002, the ACM SIGSAC Outstanding Contributions Award in 2005, the Computing Research Association Distinguished Service Award in 2013, and in 2012 was elected to the National Cybersecurity Hall of Fame as one of the first set of inductees. See his Web site, http://www.csl.sri.com/neumann, for further background and URLs for papers, reports, and testimonies. Neumann@CSL.sri.com



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

I first encountered Peter G. Neumann at the PC party for Security in Washington, DC, back around 2000. Peter was playing a grand piano and leading a group in singing songs from Gilbert and Sullivan, Tom Lehrer, and more. I later learned that Peter can play many more instruments.

Peter and I met for lunch in 2007 in Palo Alto, not far from where he works at what used to be Stanford Research Institute and is now SRI International. I was going to speak at Apple and Google over the following week about the failure of current measures that were supposed to be making our systems more secure. Peter encouraged me, then regaled me with stories about the Multics design.

Peter has been involved in security since 1965, starting with his work on the Multics file system and overall Multics development, continuing with a provably secure operating system (PSOS). His current project involves the CHERI (Capability Hardware Enhanced RISC Instructions) hardware-software system co-design [1]).

Rik Farrow: Part of what got me thinking about you was your story about part of the design of the Multics file system: getting a small group of people in a room with whiteboards and coming up with a design.

Peter G. Neumann: The first real get-together of the Multics team (MIT, Bell Labs, and GE-then-Honeywell) took place at an AT&T training center in Hopewell, NJ, the week of Memorial Day 1965. Fernando Corbató (Corby, who led the CTSS effort), Bob Daley (who created the CTSS file system), Stan Dunten (who had done the CTSS I/O), Jerry Saltzer (just about to complete his PhD thesis, "Traffic Control in a Multiplexed Computer System," 1966), and—inspirationally—Ted Glaser from MIT (co-designer with John Couleur of the really innovative hardware; former NSA, later head of the CS Department at Case Western) and his dog, and Vic Vyssotsky, Joe Ossanna, and me from Bell Labs (BTL). We discussed the emerging independently protectable segmentation hardware architecture, the desiderata for the operating system (segment descriptors, paging, and the file system), and planning for the five Fall Joint Computer Conference papers for Las Vegas in 1965. Bob Fano and Bell Labs VP Ed David (later Nixon's Science Advisor) were assigned the introductory paper, and Bob Daley and I the file system design—which largely emerged over the summer. The papers are all on multicians.org, maintained by Tom Van Vleck.

Ted was blind since age 12 but the most far-sighted person I have ever known. His impact on Multics was holistic and enormous. The first day of our week was in fact Memorial Day, and we had to find a local restaurant that was open for lunch. The only one we could find would not allow Ted's wonderful German Shepherd into the restaurant, but we finally talked them into setting up tables outside.

RF: What happened with Multics? I know Multics has continued to be used from visiting the multicians.org site, but my recollection is that the project fell apart because of disagreements between the various parties.

PGN: Bell Labs dropped out of the Multics development in 1969, when AT&T upper management realized that its declared intent that Multics would replace all computers at Murray Hill, Holmdel, Whippany, and Indian Hill could not be fulfilled on time.

Ken Thompson had joined BTL in 1967, and immediately observed that the symbolic name scheme (with dynamic linking to descriptor entries) for the file system that Bob Daley and I had designed would be great for input-output, which triggered a very nice redesign of the original Multics I/O system. As a result of Bell Labs bailing on Multics, Ken found a PDP-7 that no one was using. I remember one day when Ken came in at noon for lunch with Joe Ossanna and me, and said that he had just written a thousand-line one-user OS kernel, and I suggested he should use all of his Multics experience on multiuser multiprogramming to extend his kernel. The next day he came in with another 1000 lines. That then led to Unics (the castrated one-user Multics, so-called due to Brian Kernighan) later becoming UNIX (probably as a result of AT&T lawyers).

Multics development and maintenance continued for many years after that at MIT and at the Honeywell CISL office nearby in Cambridge. Charley Clingen headed the Honeywell Multics group, and Tom Van Vleck was heavily involved in Multics from 1966 at MIT and later moved over to Honeywell. The last Multics installation, a five-processor multiprocessor configuration, ran until 2000.

In the early 1970s there was even an effort that retrofitted multilevel security into Multics, which required a little jiggling of ring 0 and ring 1. I was a distant advisor to that (from SRI), although the heavy lifting was done by Jerry Saltzer, Mike Schroeder, and Rich Feiertag, with help from Roger Schell and Paul Karger.

The Multics hardware-software effort was seminal in pioneering Jack Dennis's notion of segmentation, with hardware-supported paging, dynamic linking, a hierarchical file system, ring structures (control hierarchies), solving the buffer overflow problem, execute-only code, pure procedure sharing, innovative file backup, and lots more. The buffer overflow problem was solved by making everything outside of the active stack frame not executable, and enforcing that in hardware.

RF: Can you tell us about your work on Provably Secure Operating System for the NSA?

PGN: Multics had a considerable influence on SRI's Provably Secure Operating System (PSOS [2]), for which the security-relevant hardware and software functionality was formally specified in a common language that we created (SpecIAL), primarily by Larry Robinson and Karl Levitt. The PSOS architecture is an early example of a hierarchically designed hardware-software system, in which each successive layer could depend only on lower layers (somewhat akin to Dijkstra's THE system [3]), but where the hardware enabled an operation at an OS or application layer to be executed efficiently as a single instruction after the descriptor table and page tables were in place. I worked on PSOS from 1973 until 1980 under a contract from the NSA. Three more years of that project supported the Goguen-Meseguer work on noninterference and early work on SRI's PVS formal verification system.

In turn, Multics and PSOS had significant influence on the CHERI that we are currently developing. In addition, the CHERI hardware supports some of the security concepts from the more recent Capsicum operating system [4])—notably, its hybrid architecture and the ability to enforce least privilege and compartmentalization.

RF: I've been reading the PSOS retrospective paper [2] and am a bit confused about what a capability is. PSOS capabilities appear associated with unique user IDs with a set of access rights. These can be copied, with restrictions, and appear to be created with hardware monotonicity that would ensure that rights could never increase.

I think I am confused because I associate capabilities with both an application and a user ID, so that a user ID doesn't have the same set of capabilities for all applications she may run. Perhaps you could explain?

PGN: You are indeed confused, perhaps because each capability system—in the past, present, and the future—tends to be slightly if not fundamentally different. PSOS capabilities were different from others because the ID of the capability was *unique* for the lifetime of that processor, and could be stored in a Multics-like directory for access via a symbolic name. There was no user ID associated, because capabilities could be shared—subject to the propagation limits. This was appropriate for the researchy hardware-software spec, as it was conceptually simple but not very practical.

CHERI capabilities [1] are more local in nature rather than global potentially for every user and every process. They are **fat** pointers, which include bounds and permissions, along with a nonforgeable tag to ensure nonforgeability of the capabilities. One common thread between PSOS and CHERI is that both have object-oriented capabilities with either default types (for virtual memory) or user-defined types (for objects).

The huge difference is that CHERI solves the legacy compatibility problem and allows crapware to coexist safely with very trustworthy operating systems, applications, compilers, and so on.

RF: The fat pointers that I know about are part of the D language extensions to C [5] and include a range with every pointer to prevent buffer overflow attacks. Can you tell me how pointers in CHERI are different?

PGN: Conventional fat pointers are typically virtual addresses that have been extended with additional metadata such as bounds and permissions. CHERI's fat-pointer capabilities add notions of sealing and unsealing (for strongly typed object capabilities), provenance (ensuring that new capabilities are properly derived from other legitimate capabilities), and monotonicity.

RF: In the PSOS paper, you describe the system as hierarchically layered, but also write that such multi-layer designs aren't found in contemporary systems. Could you explain the importance of layering and perhaps why it's not found in systems today?

PGN: Layered assurance is premised on formal analyses that can be built up layer by layer. Dijkstra's THE system [3] had informal proofs that there could be no deadlocks between layers, because the locking strategy at each layer involved purely hierarchical dependencies. Years later I asked Nico Habermann [6] about that. He said they had actually discovered a hitherto undetected deadlock within a single layer, but never any that involved multiple layers.

The Multics ring structure enabled up to eight rings, although rings 0, 1, 2 were the primary ones that were used by the system itself with ring 4 used by user software. Outer rings were left for applications. Nothing that happened in ring 1 could ever clobber ring 0, ring 2 could never clobber lower rings, and so on. Many systems have a layered structure, but typically it is only kernel and user—that is, only two layers. CHERI could implement many layers easily using the capability mechanism, either implicitly or explicitly.

PSOS had 17 layers in the conceptual architecture. Layer zero had two instructions out of which everything else was built in initialization—creating a new capability with desired privileges and creating a copy of an existing capability with at most the same privileges. That's CHERI's monotonicity property (which includes privileges and bounds that may never increase). The lowest 7 PSOS layers were intended to be implemented in hardware.

The Multics ring property is conceptually similar to the Biba multilevel integrity dependence property—that each layer (or in Biba's case, integrity level) must depend only on itself and on lower layers, at least in principle. There are of course some trusted exceptions involving calling into a lower ring—and then returning without acquiring any lower-layer privileges. CHERI explicitly introduces the principle of intentionality to counter the fact that calling something else must not allow the something else to confer properties elsewhere or usurp privileges it does not have. This addresses so-called confused-deputy attacks.

As you can see, this all fits together—from Multics to PSOS and Capsicum to CHERI, with deep awareness from my Cambridge colleagues on all of the other attempts at past and contemporaneous capability-based systems, and proactively trying to avoid the pitfalls of the past while adopting other ideas that might work in this context (such as capabilities that act as fat pointers). Robert Watson in particular has an absolutely uncanny understanding of all of this, and someone without whom we could have never gotten this far so quickly in developing CHERI.

RF: This is making sense to me. I didn't realize that the lower seven layers of PSOS were supposed to be done in hardware.

I'm glad you brought up rings, as people widely misunderstand them today. I did want to mention that virtualization actually has meant the creation of more rings, such as "ring -1" for hypervisors.

What I was wondering is whether CHERI provides a model for capabilities that would be useful for people to learn about today? I haven't finished reading the technical report yet, but it seems like capabilities are a bit like container technology, in that capabilities are used to control access to various namespaces, very much like Linux containers.

How closely does CHERI mimic the systems that Multics ran on? I realize that both the GE 645 and CHERI make use of segment registers as hardware support for isolation. That seems different from the capabilities discussed in Capsicum.

PGN: PSOS used ideas from Multics. CHERI used ideas both from Multics and from PSOS and Capsicum. But Capsicum is software only and relies on potentially untrustworthy hardware. We rectified that in CHERI, which adopted the hybrid model of Capsicum, but designed hardware that would greatly enhance the trustworthiness of operating systems and applications. It also advances operating systems beyond Capsicum. Try to understand the CHERI papers [7] as new stuff, although each paper states how we differ from the past. The tech report will help a lot. The report is long but well structured. It includes a chronological history of how we got to where we are, as well as how it relates to other efforts. All of this should be extraordinarily valuable for learning about the security pitfalls that can be overcome through enlightened hardware and total-system architecture.

RF: Does CHERI provide the same or better support in hardware than did systems running Multics? Does CHERI's hardware support extend beyond segment registers, for hardware that provides real isolation for different capabilities? Those are questions I'd like you to answer.

PGN: The Multics hardware was designed by John Couleur and Ted Glaser. John was a pure hardware person. Ted was someone who got John to build independently protectable segmentation into the hardware, with a deep understanding of how the operating system and compilers might exploit it for paging and shared pure procedure, as well as for security, reliability, robustness, resilience, and more.

The CHERI hardware ISA began with an open-source MIPS 64-bit ISA formal spec (developed by Cambridge), and added capability instructions and capability registers. It represents a complete clean-slate hardware-software co-design. CHERI has proceeded iteratively, with a few very minor but useful additions or refinements of particular instructions over the past seven years because of better understanding of the operating-system and compiler needs.

CHERI can do anything Multics could do—segmentation, paging, dynamic linking, ring-structured software—and much more (high-assurance fine-grained access controls, fine- and coarse-grained compartmentalization, e.g., within a given application or within an OS, and among all of the different applications, virtual partitions and more). We believe that we will soon have some viable approaches to the active device input-output direct memory access problems. The Multics General Input/Output Controller (GIOC) had that problem in spades, because the GIOC needed absolute memory addresses, bypassing all segmentation, paging, and memory protection. CHERI hopes to extend the reach of the capability-based protection to I/O and embedded active devices and microcontrollers.

A big difference between Multics and CHERI development is that the Honeywell 645 was pretty much frozen early in the hardware design. Getting the operating-system dynamic linking to work with the hardware took several iterations, and might have been abetted by hardware improvements that were not available. On the other hand, the CHERI ISA has been fluid and able to respond to the needs of software and compilers, as we increasingly learned how to take advantage of the CHERI capability architecture—which is somewhat different from most of the predecessor capability systems. Various instructions were added along the way to simplify software development. CHERI also adopted the PSOS idea of capabilities for typed objects in hardware (noted above), which was not possible in Multics.

There is considerable detail that we have glossed over, and other efforts such as microkernel operating systems, application trustworthiness, and the use of formal methods to ensure that the hardware ISA satisfies the required trustworthiness properties and principles. In addition to [1], see [8] and [9].

References

[1] R. N. M. Watson, R. Norton, J. Woodruff, A. Joannou, S. W. Moore, P. G. Neumann, J. Anderson, D. Chisnall, N. Dave, B. Davis, K. Gudka, B. Laurie, A. T. Markettos, E. Maste, S. J. Murdoch, M. Roe, C. Rothwell, S. Son, and M. Vadera, "Fast Protection-Domain Crossing in the CHERI Capability-System Architecture," *IEEE Micro Journal*, vol. 36, no. 6 (September-October 2016), pp. 38–49: https://goo.gl/Vu8W1J.

The current comprehensive hardware ISA document is online: R. N. M. Watson, P. G. Neumann, J. Woodruff, M. Roe, J. Anderson, J. Baldwin, D. Chisnall, B. Davis, B. Laurie, S. W. Moore, S. J. Murdoch, R. Norton, S. Son, H. Xia, "Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 6)," Technical Report no. 907, University of Cambridge Computer Laboratory, July 2017: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-907.pdf.

[2] P. G. Neumann and R. Feiertag, "PSOS, Revisited," ACSAC, 2003: http://www.csl.sri.com/users/neumann/psos03.pdf; http://www.csl.sri.com/neumann/psos/psos80.pdf (full 1980 report—scanned).

[3] E.W. Dijkstra, "The Structure of the THE Multiprogramming System," *Communications of the ACM*, vol. 11, no. 5 (May 1968), pp. 341–346; also, Wikipedia, "Edsger W. Dijkstra," section 2.5 (Operating system research): https://en.wikipedia.org/wiki/Edsger_W._Dijkstra#Operating_system_research.

[4] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical Capabilities for Unix," in *Proceedings of the 19th USENIX Security Symposium*, August 2010; see also Capsicum Technologies: https://wiki.freebsd.org/Capsicum.

[5] Cello, "A Fat Pointer Library": http://libcello.org/learn/a-fat-pointer-library.

[6] Wikipedia, "Nico Habermann": https://en.wikipedia.org/wiki/Nico_Habermann.

[7] CHERI project home page: http://www.cl.cam.ac.uk/research/security/ctsrd/cheri/.

[8] R. N. M. Watson, P. G. Neumann, and S. W. Moore, "Balancing Disruption and Deployability in the CHERI Instruction-Set Architecture (ISA)," in *New Solutions for Cybersecurity*, ed. H. Shrobe, D. Shrier, A. Pentland (MIT Press/Connection Science, 2018).

[9] P. G. Neumann, "Fundamental Trustworthiness Principles," in *New Solutions for Cybersecurity*, ed. H. Shrobe, D. Shrier, A. Pentland (MIT Press/Connection Science, 2018).

Save the Date!

27TH USENIX SECURITY SYMPOSIUM

August 15–17, 2018 • Baltimore, MD, USA

The USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security and privacy of computer systems and networks.

Submit your work!
Submissions are due February 8, 2018.

Program Co-Chairs
William Enck, *North Carolina State University,*
and Adrienne Porter Felt, *Google*

www.usenix.org/sec18

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Save the Date!

Fourteenth Symposium on Usable Privacy and Security

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

Submit your work!
Abstract submissions are due February 12, 2018.
Full paper submissions are due February 16, 2018.

Symposium Organizers

General Chair
Mary Ellen Zurko, *MIT Lincoln Laboratory*

Vice General Chair
Heather Richter Lipford,
University of North Carolina at Charlotte

Technical Papers Co-Chairs
Sonia Chiasson, *Carleton University*
Rob Reeder, *Google*

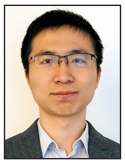
www.usenix.org/soups2018

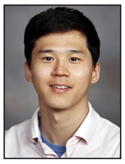
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

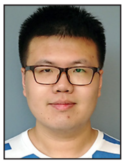
SYSTEMS

Decentralized Memory Disaggregation Over Low-Latency Networks

JUNCHENG GU, YOUNGMOON LEE, YIWEN ZHANG, MOSHARAF CHOWDHURY, AND KANG G. SHIN


Juncheng Gu is a PhD student at the University of Michigan. He is broadly interested in systems and networks. His current focus is resource disaggregation using RDMA networks. jcgu@umich.edu


Youngmoon Lee is a PhD candidate at the University of Michigan. He works on cloud and mobile systems, and his current research focuses on resilient cloud computing and resource management. ymoonlee@umich.edu


Yiwen Zhang is a master's student at the University of Michigan. His research interests include computer networks and RDMA performance isolation. yiwenzhg@umich.edu

Mosharaf Chowdhury is an Assistant Professor in the EECS Department at the University of Michigan. His research ranges from resource disaggregation in low-latency RDMA networks to geo-distributed analytics over the WAN, with a common theme of enabling application-infrastructure symbiosis across different layers of corresponding software and hardware stacks. mosharaf@umich.edu

Memory disaggregation can expose remote memory across a cluster to local applications. However, existing proposals call for new architectures and/or new programming models, making them infeasible. We have developed a practical memory disaggregation solution, Infiniswap, which is a remote memory paging system for clusters with low-latency, kernel-bypass networks such as RDMA. Infiniswap opportunistically harvests and transparently exposes unused memory across the cluster to unmodified applications by dividing the swap space of each machine into many chunks and distributing them to unused memory of many remote machines. For scalability, it leverages the power of many choices to perform decentralized memory chunk placements and evictions. Applications using Infiniswap receive large performance boosts when their working sets are larger than their physical memory allocations.

Motivation

Modern operating systems (OSes) provide each application with a virtual memory address space that is much larger than its physical memory allocation. Whenever an application addresses a virtual address whose corresponding virtual page does not reside in the physical memory, a *page fault* is raised. If there is not enough space in the physical memory for that virtual page, the virtual memory manager (VMM) may need to *page out* one or more in-memory pages to a block device, which is known as the *swap space*. Subsequently, the VMM brings the missing page into the physical memory from the swap space; this is known as *paging in*.

Performance Degradation from Paging

Due to the limited performance of traditional swap spaces—typically, rotational hard disks—paging in and out can significantly affect application performance. To illustrate this issue, we select four memory-intensive applications: (1) a standard TPC-C benchmark running on the VoltDB in-memory database; (2) two Facebook-like workloads running on the Memcached key-value store; (3) the TunkRank algorithm running on PowerGraph with a Twitter data set; and (4) GraphX running the PageRank algorithm in Apache Spark using the same Twitter data set.

We run each application in its own container with different memory constraints. *x%* in the X-axes of Figure 1 refers to a run inside a container that can hold at most *x%* of the application's working set in memory, and ×< 100 forces paging in from/out to the machine's *swap space*.

Figure 1 shows significant, non-linear impact on application performance due to paging. For example, a 25% reduction of memory results in a 5.5× and 2.1× throughput loss for VoltDB and Memcached, respectively; PowerGraph and GraphX worsen marginally. However, another 25% reduction makes VoltDB, Memcached, PowerGraph, and GraphX up to 24×, 17×, 8×, and 23× worse, respectively. These gigantic performance degradations reflect the potential benefits that an efficient memory disaggregation system can deliver.

Kang G. Shin is the Kevin & Nancy O'Connor Professor of Computer Science in the Department of Electrical Engineering and Computer Science, University of Michigan. His current research focuses on QoS-sensitive computing and networking as well as on embedded real-time and cyber-physical systems. He has supervised the completion of 80 PhDs and has authored/co-authored more than 900 technical articles, a textbook, and more than 30 patents or invention disclosures; he has received numerous best paper awards. He was a co-founder of a couple of startups and also licensed some of his technologies to industry. kgshin@umich.edu

Characteristics of Memory Imbalance

Memory utilization is imbalanced across machines in a cluster. Although some machines are under heavy memory pressure, others in the same cluster can still have unused memory. Causes of imbalance include placement and scheduling constraints [3, 4] and resource fragmentation during packing [8]. To understand the presence of memory imbalance in clusters and corresponding opportunities, we analyzed traces from two production clusters: (1) a 3000-machine data analytics cluster (Facebook) and (2) a 12,500-machine cluster (Google) running a mix of diverse short- and long-running applications.

Presence of Imbalance. We measured memory utilization imbalance by calculating the 99th-percentile to the median usage ratio over 10-second intervals (Figure 2). With a perfect balance, these values would be 1. However, we found this ratio to be 2.40 in Facebook and 3.35 in Google more than half the time; meaning, most of the time, more than half of the cluster aggregate memory remains unutilized.

Temporal Variabilities. Although skewed, memory utilizations remained stable over short intervals, which is useful for predictable decision-making when selecting remote machines. We observed that average memory utilizations of a machine remained stable for smaller durations with very high probabilities. For the most unpredictable machine in the Facebook cluster, the probabilities that its current memory utilization from any instant will not change by more than 10% for the next 10, 20, and 40 seconds were 0.74, 0.58, and 0.42, respectively. For Google, the corresponding numbers were 0.97, 0.94, and 0.89, respectively.

The presence of memory imbalance and its temporal variabilities suggest opportunities for harvesting unused memory across a cluster by memory disaggregation.

Infiniswap Overview

Infiniswap is a decentralized memory disaggregation solution for clusters with low-latency, kernel-bypass networks such as RDMA. The main goal of it is to *efficiently expose all of a cluster's memory to user applications*. To avoid modifying existing applications or OSes, Infiniswap provides remote memory to local applications through the already-existing paging mechanism.

Infiniswap has two primary components—Infiniswap block device and Infiniswap daemon—that are present in every machine and work together without any central coordination (Figure 3).

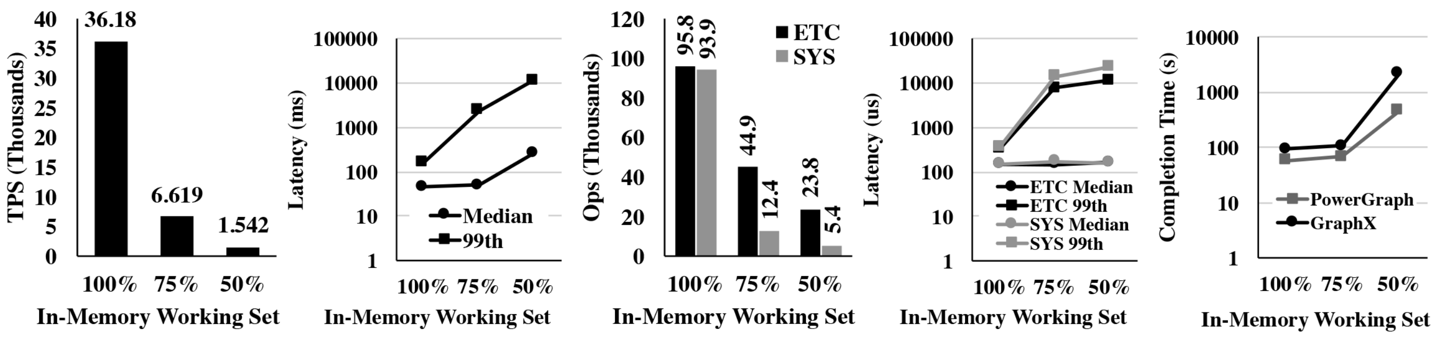


Figure 1: For modern in-memory applications, a decrease in the percentage of the working set that fits in memory often results in a disproportionately larger loss of performance. This effect is further amplified for tail latencies. All plots show single-machine performance and the median value of five runs. Lower is better for the latency-related plots (lines), and the opposite holds for the ones (bars). Note the logarithmic Y-axes in the throughout-related latency/completion time plots.

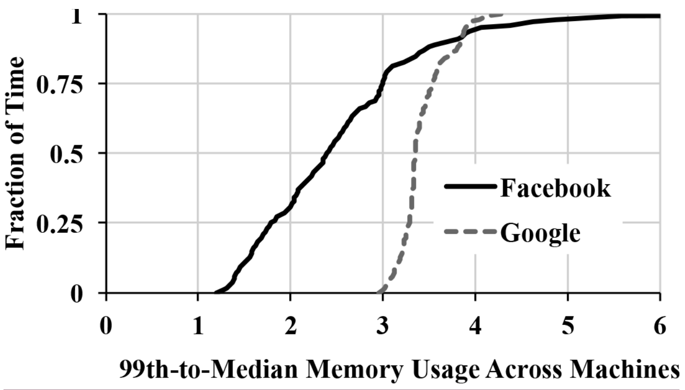


Figure 2: Imbalance in 10s-averaged memory usage in two large production clusters at Facebook and Google

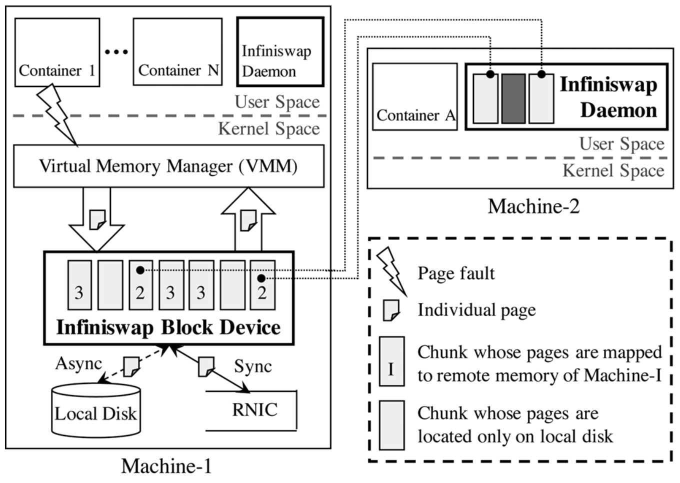


Figure 3: Infiniswap architecture. Each machine loads a block device as a kernel module (set as swap device) and runs an Infiniswap daemon. The block device divides its storage space into chunks and transparently maps them across many machines' remote memory; paging happens at page granularity via RDMA.

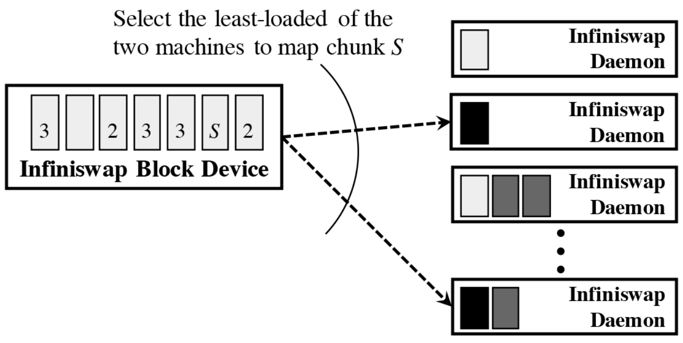


Figure 4: Infiniswap block device uses power of two choices to select machines with the most available memory. It prefers machines without any of its chunks over those that have chunks. In this way, its chunks can be distributed across as many machines as possible.

The Infiniswap block device exposes a conventional block device I/O interface to the virtual memory manager (VMM), which treats it as a fixed-size swap partition. The entire storage space of this device is logically partitioned into fixed-size *chunks* ("ChunkSize"). A chunk represents a contiguous region, and it is the unit of remote mapping and load balancing in Infiniswap. Chunks from the same block device can be mapped to multiple remote machines' memory for load balancing. The VMM stores and retrieves data from the Infiniswap block device at page granularity. All pages belonging to the same chunk are mapped to the same remote machine. On the Infiniswap daemon side, a chunk is a physical memory region of ChunkSize that is mapped to and used by an Infiniswap block device as remote memory.

Infiniswap consults the status of remote memory mapping to handle paging requests. If a chunk is mapped to remote memory, Infiniswap synchronously writes a page-out request for that chunk to remote memory using `RDMA WRITE`, while writing it asynchronously to the local disk. If it is not mapped, Infiniswap synchronously writes the page only to the local disk. For page-in requests, Infiniswap reads data from the appropriate source; it uses `RDMA READ` for remote memory.

The Infiniswap daemon only participates in control plane activities. It (1) responds to chunk-mapping requests from Infiniswap block devices; (2) pre-allocates its local memory when possible to minimize time overheads in chunk-mapping initialization; and (3) proactively evicts chunks, when necessary, to ensure minimal impact on local applications. All control plane communications take place using `RDMA SEND/RECV`.

Scalability. Infiniswap leverages the well-known power of choices techniques [6, 7] during both chunk placement and eviction. The reliance on decentralized techniques makes Infiniswap more scalable by avoiding the need for constant coordination, while still achieving low-latency mapping and eviction.

Fault Tolerance. With the decentralized approach, Infiniswap does not have a single point of failure. It considers unreachability of remote daemons (e.g., due to machine failures, daemon process crashes, etc.) as the primary failure scenario. If a remote daemon becomes unreachable, the Infiniswap block device relies on the remaining remote memory and the local backup disk. If the local disk also fails, Infiniswap provides the same failure semantic as of today.

Efficient Memory Disaggregation via Infiniswap Block Device

An Infiniswap block device logically divides its entire storage space into multiple chunks of fixed size (ChunkSize). Using a fixed size throughout the cluster simplifies chunk placement and eviction algorithms and their analyses.

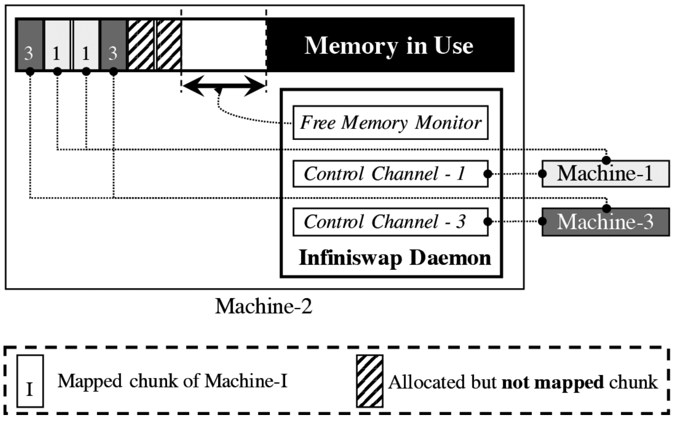


Figure 5: The Infiniswap daemon periodically monitors available free memory to pre-allocate chunks and to perform fast evictions. Each machine runs one daemon.

Remote Chunk Placement

Each chunk starts in the *unmapped* state. Infiniswap monitors the paging activity rates of each chunk using an exponentially weighted moving average (EWMA). When the paging activity of an unmapped chunk crosses the HotChunk threshold, Infiniswap attempts to map that chunk to a remote machine's memory.

The chunk placement algorithm has the following goals. First, it should *distribute chunks from the same block device* across as many remote machines as possible in order to minimize the impacts of future evictions from (or failures of) remote machines. Second, it attempts to *balance memory utilization* across all the machines to minimize the probability of future evictions. Finally, it must be *decentralized* to provide low-latency mapping without central coordination.

Instead of randomly selecting an Infiniswap daemon without central coordination, we leverage the power of two choices [6] to minimize memory imbalance across machines (Figure 4). First, Infiniswap divides all the machines (M) into two sets: those that already have any chunk of this block device (M_{old}) and those that do not (M_{new}). Next, it contacts two Infiniswap daemons and selects the one with the lowest memory usage. It first selects from M_{new} and then, if necessary, from M_{old}. The two-step combination distributes chunks across many machines while decreasing load imbalance in a decentralized manner.

Handling Chunk Evictions and Remote Failures

Upon receiving an eviction message from the Infiniswap daemon, the Infiniswap block device marks the chunk as unmapped. All future requests of the unmapped chunk will go to disk. The Infiniswap block device cannot send the eviction response back to the Infiniswap daemon until all the in-flight requests of that chunk are completed. The workflow of handling remote failures

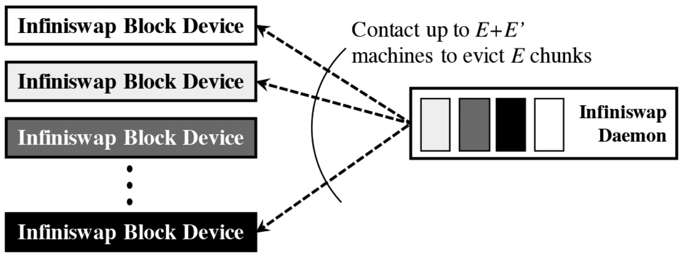


Figure 6: The Infiniswap daemon employs batch eviction (i.e., contacting *E´* more chunks to evict *E* chunks) for fast eviction of *E* lightly active chunks.

is similar to that of chunk eviction: mark the affected chunk(s) as unmapped, and forward future requests to disk.

Transparent Remote Memory Reclamation via Infiniswap Daemon

The core functionality of each Infiniswap daemon is to claim memory on behalf of remote block devices as well as reclaiming them on behalf of the applications on its host.

Memory Management

The Infiniswap daemon periodically monitors the total memory usage of everything else running on its host. In order to be transparent to applications on the same machine, it focuses on maintaining a "HeadRoom" amount of free memory in the machine by controlling its own total memory allocation. The optimal value of "HeadRoom" should be dynamically determined based on the amount of memory and the applications running in each machine. Our current implementation does not include this optimization and uses 8-GB "HeadRoom" by default on 64-GB machines.

When the amount of free memory grows above "HeadRoom," the Infiniswap daemon proactively allocates chunks of size ChunkSize and marks them as unmapped. Proactive allocation of chunks makes the initialization process faster when an Infiniswap block device attempts to map to that chunk; the chunk is marked mapped at that point.

When free memory shrinks below "HeadRoom," the Infiniswap daemon proactively releases chunks in two stages. It starts by releasing unmapped chunks. Then, if necessary, it *evicts E* mapped chunks.

Decentralized Chunk Eviction

To minimize the performance impact on the Infiniswap block devices that are remotely mapped, the Infiniswap daemon should select the least-active mapped chunks for eviction. The key challenge arises from the one-sided `RDMA (READ/WRITE)` operations used in the data plane of Infiniswap. While this avoids CPU involvements, it also prevents the Infiniswap

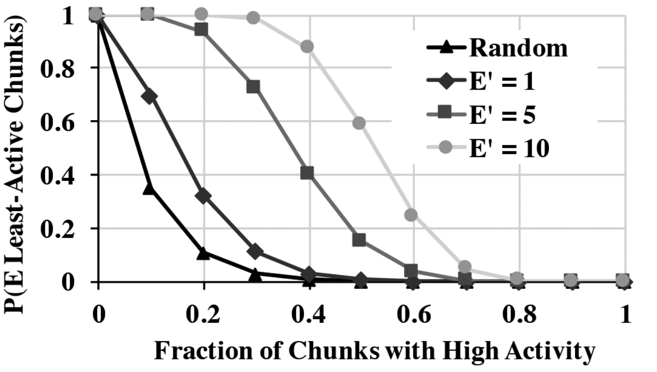


Figure 7: Analytical eviction performance for evicting *E(= 10)* chunks for varying values of *E´*. Random refers to evicting *E* chunks one by one uniformly randomly.

daemon from gathering any paging activities of the mapped chunks without first communicating with the corresponding block devices.

Consider a scenario where a daemon needs to release *E* mapped chunks. At one extreme, the solution is to collect global knowledge by contacting *all* related block devices to determine the least-used *E* chunks. This is prohibitive when *E* is significantly smaller than the total number of mapped chunks. Having a centralized controller would not have helped either, because this would require *all* Infiniswap block devices to frequently report their chunk activities.

At the other extreme, one can randomly pick one chunk at a time without any communication. However, in this case, the likelihood of evicting a busy chunk is very high. Consider a parameter $p_b \in [0, 1]$, and assume that a chunk is busy (i.e., it is experiencing paging activities beyond a fixed threshold) with probability p_b. The probability of finding *E* lightly active chunks would be $(1 - p_b)^E$. As the cluster becomes busier (p_b increases), this probability plummets (Figure 7).

Batch Eviction. Instead of randomly evicting chunks without any communication, we perform bounded communication to leverage generalized power of choices [7].

For *E* chunks to evict, the Infiniswap daemon considers *E* + *E'* chunks, where *E'* ≤ *E*. Upon communicating with the Infiniswap block devices of those *E* + *E'* chunks, it evicts *E* least-active ones. The probability of finding *E* lightly active chunks in this case is

$$\sum_{E}^{E+E'} (1 - p_b)^i {p_b}^{E+E'-i} \binom{E+E'}{i}$$

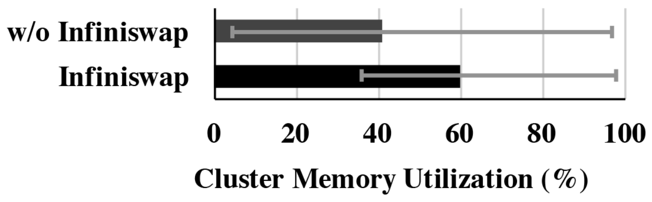


Figure 8: Using Infiniswap, memory utilization increases and memory imbalance decreases significantly. Error bars show the maximum and the minimum utilization across machines.

Figure 7 plots the effectiveness of batch eviction for different values of *E'* for *E* = 10. Even for moderate cluster load, the probability of evicting lightly active chunks is significantly higher using batch eviction.

Implementation

We have implemented Infiniswap as a loadable kernel module for Linux 3.13.0 and beyond in about 3500 lines of C code. Our block device implementation is based on nbdX [1], a network block device over Accelio framework, developed by Mellanox. Infiniswap daemons are implemented and run as userspace programs. More implementation details can be found in our NSDI paper [5].

Evaluation

We evaluated Infiniswap on a 32-machine, 56 Gbps Infiniband cluster on CloudLab [2] and highlight two key results as follows:

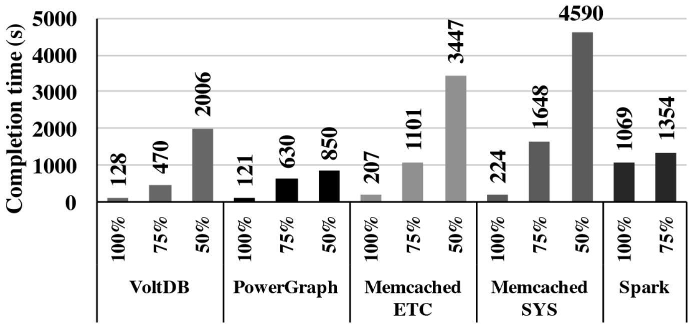
- In comparison to traditional swap spaces such as rotational disks, Infiniswap improves throughputs of unmodified VoltDB, Memcached, PowerGraph, GraphX, and Apache Spark from 4× to up to 15.4× and tail latencies by up to 61×.
- Infiniswap benefits hold in a distributed setting. It increases cluster memory utilization by 1.47× using a small amount of network bandwidth.

The rest of this section describes how Infiniswap performs in a cluster with many applications. Details about our experimental setup, workload configurations, and more evaluation results can be found in our NSDI paper [5].

Cluster-Wide Performance

Methodology

We used the same application-workload combinations in Figure 1 to create about 90 containers. Each combination has an equal number of containers. About 50% of them had no memory constraint, close to 30% used the 75% memory constraint, and the rest used the 50% memory constraint. They were placed randomly across 32 machines to create a memory imbalance scenario similar to those shown in Figure 2.



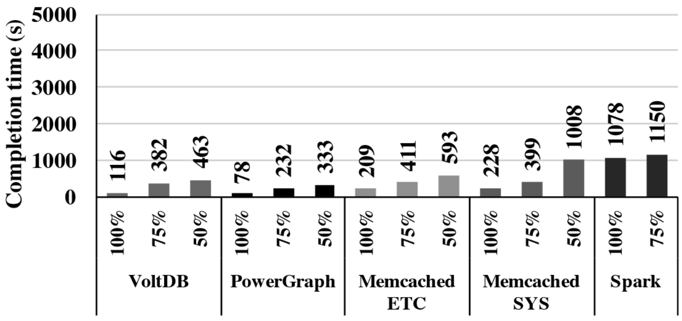


Figure 9: Median completion times of containers for different configurations in the cluster experiment. Infiniswap's benefits translate well to a larger scale in the presence of high application concurrency.

Cluster Memory Utilization
Infiniswap improves total cluster memory utilization by 1.47× by increasing it to 60% on average from 40.8% (Figure 8). Moreover, Infiniswap significantly decreases memory imbalance: the maximum-to-median utilization ratio decreased from 2.36× to 1.60×, and the maximum-to-minimum utilization ratio decreased from 22.5× to 2.7×.

Application-Level Performance
We observe that Infiniswap holds its benefits in the presence of cluster dynamics of many applications (Figure 9). Although improvements are sometimes lower than those observed in controlled single-instance scenarios [5], Infiniswap still provides 3×–6× improvements over disk for the 50% memory constraint.

Ongoing Efforts
We are actively extending Infiniswap in two directions:

Fault Tolerance
Infiniswap can tolerate the failures of remote machines with its backup disk. However, backing up data on hard disk becomes the performance bottleneck of the entire system when many swap bursts come together. We are considering trying to achieve the fault tolerance feature by distributing data to multiple remote machines using erasure coding.

Performance Isolation
Infiniswap provides remote memory to all the applications running on the machine. As such, it cannot distinguish between pages from specific applications. Swap requests originating from different applications share the same resources in Infiniswap, such as dispatch buffers in Infiniswap and cache on RDMA NICs. Consequently, Infiniswap cannot guarantee performance isolation among multiple applications on the same host.

Conclusion
Infiniswap is a pragmatic solution for memory disaggregation without requiring any modifications to applications, OSes, or hardware. It bypasses CPU through one-sided RDMA operations in the data plane for performance, and it uses scalable, decentralized remote memory placement and eviction schemes in the control plane for fault tolerance and scalability. We have demonstrated Infiniswap's advantages in substantially improving the performance of multiple popular memory-intensive applications. Infiniswap also increases the overall memory utilization of a cluster, and its benefits hold at scale.

The source code of Infiniswap and more information are available at https://infiniswap.github.io/infiniswap/.

Acknowledgments
Special thanks go to the entire CloudLab team—especially Robert Ricci, Leigh Stoller, and Gary Wong—for pooling together enough resources to make Infiniswap experiments possible. We would also like to thank the anonymous reviewers and our shepherd, Mike Dahlin, for their insightful comments and feedback that helped improve the paper. This work was supported in part by National Science Foundation grants CCF-1629397, CNS-1563095, CNS-1617773, by the ONR grant N00014-15-1-2163, and by an Intel grant on low-latency storage systems.

References

[1] Accelio-based network block device: https://github.com/accelio/NBDX.

[2] CloudLab: https://www.cloudlab.us.

[3] P. Bodik, I. Menache, M. Chowdhury, P. Mani, D. Maltz, and I. Stoica, "Surviving Failures in Bandwidth-Constrained Datacenters," in *Proceedings of the ACM Conference on Data Communication (SIGCOMM '12)*, pp. 431–442: http://bit.ly/2xDVSOs.

[4] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging Endpoint Flexibility in Data-Intensive Clusters," in *Proceedings of the ACM Conference on Data Communication (SIGCOMM '13)*, pp. 231–242: http://bit.ly/2fqyzgU.

[5] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient Memory Disaggregation with Infiniswap," i n *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*: http://bit.ly/2yFiMDl.

[6] M. Mitzenmacher, A. W. Richa, and R. Sitaraman, "The Power of Two Random Choices: A Survey of Techniques and Results," *Handbook of Randomized Computing* (Springer, 2001), pp. 255–312.

[7] G. Park, "Brief Announcement: A Generalization of Multiple Choice Balls-into-Bins," in *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC '11)*, pp. 297–298.

[8] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-Scale Cluster Management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys '15)*: http://bit.ly/2ye4ecV.

Save the Date!

FAST '18

16th USENIX Conference on File and Storage Technologies

February 12–15, 2018 • Oakland, CA, USA

FAST '18 brings together storage-system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems. The program committee will interpret "storage systems" broadly; everything from low-level storage devices to information management is of interest. The conference will consist of technical presentations, including refereed papers, Work-in-Progress (WiP) reports, poster sessions, and tutorials.

The full program and registration will be available in December 2017.

www.usenix.org/fast18

usenix® THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Save the Date!

nsdi '18

15th USENIX Symposium on Networked Systems Design and Implementation

April 9–11, 2018 • Renton, WA, USA

NSDI '18 focuses on the design principles, implementation, and practical evaluation of networked and distributed systems. Our goal is to bring together researchers from across the networking and systems community to foster a broad approach to addressing overlapping research challenges.

The full program and registration will be available in January 2018.

www.usenix.org/nsdi18

usenix® THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Psychological Safety in Operation Teams

JOHN P. LOONEY

John Looney is a Systems Engineer at Intercom, helping to build a modern SaaS-based infrastructure platform. Before that, he was a full-stack SRE at Google, where he did everything from rack design and datacenter automation to ad-serving; he had stops at GFS, Borg, and Colossus along the way. He wrote a chapter of the SRE book on automation and is on the steering committee for USENIX SREcon. valen@tuatha.org

When I worked for Google as a Site Reliability Engineer, I was lucky enough to travel around the world with a group called "Team Development." Our mission was to design and deliver team-building courses to teams who wanted to work better together. Our work was based on research later published as Project Aristotle [1]. It found that the primary indicator of a successful team wasn't tenure, seniority, or salary levels but psychological safety.

Think of a team you work with closely. How strongly do you agree with these five statements?

1. If I take a chance and screw up, it will be held against me.
2. Our team has a strong sense of culture that can be hard for new people to join.
3. My team is slow to offer help to people who are struggling.
4. Using my unique skills and talents comes second to the objectives of the team.
5. It's uncomfortable to have open, honest conversations about our team's sensitive issues.

Teams that score high on questions like these can be deemed to be "unsafe." Unsafe to innovate, unsafe to resolve conflict, unsafe to admit they need help. Unsafe teams can deliver for short periods of time, provided they can focus on goals and ignore interpersonal problems. Eventually, unsafe teams will underperform or shatter because they resist change.

Let me highlight the impact an unsafe team can have on an individual, through the eyes of an imaginary, capable, and enthusiastic new college graduate.

This imaginary graduate, I'll call her Karen, read about a low-level locking optimization for distributed databases and realized it applied to the service her team was on-call for. Test results showed a 15% CPU saving! She excitedly rolled it out to production. Changes to the database configuration file didn't go through the usual code-review process, and, unfortunately, it caused the database to hard-lock-up. There was a brief but total Web site outage. Thankfully, her more experienced colleagues spotted the problem and rolled back the change inside of 10 minutes. Being professionals, the incident was discussed at the weekly "postmortem" meeting.

1. "If I take a chance, and screw up, it'll be held against me."

At the meeting, the engineering director asserted that causing downtime by chasing small optimizations was unacceptable. Karen was described as "irresponsible" in front of the team. The team suggested ways to ensure it wouldn't happen again. Unlike Karen, the director soon forgot about this interaction.

Karen would never try to innovate without explicit permission again.

2. "Our team has a strong sense of culture, and it's hard for new people to join."

The impact on Karen was magnified because no one stood up for her. No one pointed out the lack of code reviews on the database configuration. No one highlighted the difference between one irresponsible act and labeling someone "irresponsible." The team was proud of their system's reliability, so defending their reputation was more important than defending a new hire.

Karen learned that her team and manager didn't have her back.

3. "My team is slow to offer help to people who are struggling."

Karen was new to being on-call for a "production" system, so had no formal training in incident management, production hygiene, or troubleshooting distributed systems. Her team was mostly made up of people with decades of experience, who never needed training or new-hire documentation. There were no signals that it was OK for a new graduate to spend time learning these skills.

Karen was terrified of being left with the pager. She didn't understand how she passed the hiring process, and frequently wondered why she hadn't been fired yet. We call this Imposter Syndrome [2].

4. "Using my unique skills and talents comes second to the goals of the team."

Karen's background was in algorithms, data structures, and distributed computing. She realized the existing system had design flaws and could never handle load spikes. The team had always blamed the customers for going over their contracted rates, which is like blaming weathermen for rain during an Irish barbecue. Strong operations teams need a mix of people from different backgrounds. It's not always clear whether a problem will require understanding a database schema, Ruby debugging, C++ performance understanding, product knowledge, or people skills.

Karen proposed a new design, based on technology she'd used during her internship. Her coworkers were unfamiliar with the new technology and considered it too risky. Karen dropped her proposal without discussion. She wanted to write code and build systems, not have pointless arguments.

5. "It's uncomfortable to have open, honest conversations about our team's sensitive issues."

When a large customer traffic spike caused the product to be unavailable for a number of hours, the CEO demanded a meeting with the operations team. Many details were discussed, and Karen explained that the existing design meant it could never deal with such spikes and mentioned her design. Her director reminded her that her design had already been turned down at an Engineering Review and promised the CEO they could improve the existing design.

Karen discussed the meeting with one of her teammates afterwards. She expressed dismay that the director couldn't see that his design was the root-cause of their problems. The teammate shrugged and pointed out that the team had delivered a really good service for the last five years and had no interest in arguing about alternate designs with the director.

Karen left work early to look for a new job. The company didn't miss her when she left. After all, she was "reckless, whiny and had a problem with authority." They didn't reflect on the design that would have saved the company from repeated outages that caused a customer exodus.

How to Build Psychological Safety into Your Own Team

What is special about Operations that drives away so many promising engineers and suffers others to achieve less than their potential?

We know that success requires a strong sense of culture, shared understandings and common values. We have to balance that respect for our culture with an openness to change it as needed. A team—initially happy to work from home—needs to co-locate if they take on interns. Teams—proud that every engineer is on-call for their service—may need to professionalize around a smaller team of operations-focused engineers as the potential production impact of an outage grows.

We need to be thoughtful about how we balance work people love with work the company needs to get done. Good managers are proactive about transferring out an engineer who is a poor fit for their team's workload. Great managers expand their team's remit to make better use of the engineers they have, so they feel their skills and talents are valued. Engineers whose skills go unused grow frustrated. Engineers ill-equipped to succeed at assigned work will feel set up to fail.

Make Respect Part of Your Team's Culture

It's hard to give 100% if you spend mental energy pretending to be someone else. We need to make sure people can be themselves by ensuring we say something when we witness disrespect. David Morrison (Australia's Chief of the Army) captured this sentiment perfectly in his "the standard you walk past is the standard you accept" [3] speech. Being thoughtless about people's feelings and experiences can shut them down. Some examples where I've personally intervened:

- Someone welcomes a new female project manager to the team, assumes they aren't technical, and uses baby words to explain a service. I highlight the new PM has a PhD in CS. No harm was intended, and the speaker was mortified that their good-humored introduction was inappropriate.
- In a conversation about people's previous positions, someone mentioned they worked for a no-longer-successful company, and a teammate mocked them for being "brave enough" to admit it. I pointed out that mocking people is unprofessional and unwelcome, and everyone present understood a "line" that hadn't been visible previously.
- A quiet, bright engineer consistently gets talked over by extroverts in meetings. I point out to the "loud" people that we were missing an important viewpoint by not ensuring everyone speaks up. Everyone becomes more self-aware.

It's essential to challenge lack of respect immediately, politely, and in front of everyone who heard the disrespect. It would have been wonderful had someone reminded Karen's director, in front of the group, that Karen wasn't irresponsible, the outage wasn't a big deal, and the team should improve their test coverage.

Make Space for People to Take Chances

Some companies talk of 20% time. Intercom, where I work, has "buffer" weeks, in between some of our six-week sprints [4]. People often take that chance to scratch an itch that was bothering them, without impacting the external commitments the team has made. Creating an expectation that everyone on the team has permission to innovate, and encouraging the whole team to go off-piste at the same time, sends a powerful message.

Be careful that "innovation time" isn't the only time people should take chances. I've worked with one company in the car industry that considers "innovation time" to be 2:30 p.m. on Tuesdays!

Imagine how grateful Karen would have been had a senior engineer at the Engineering Review offered to work on her design with her so that it was more acceptable to the team. Improve people's ideas rather than discounting them.

Make It Obvious When Your Team Is Doing Well

One engineer describes his experience of on-call as "being like the maintenance crew at the fairground. No one notices our work, until there is a horrible accident." Make sure people notice when your team is succeeding.

I love how my team writes goals on Post-It notes at our daily standups and weekly goal meetings. These visible marks of success can be cheered as they are moved to the "done" pile. But we can also celebrate glorious failure.

Many years ago, when I was running one of Google's storage SRE teams, we were halfway through a three-year project to replace the old Google File System. Through a confluence of bad batteries, firmware bugs, poor tooling, untested software, an aggressive rollout schedule, and two power cuts, we lost a whole storage cell for a number of hours. Though all services would have had storage in other availability zones, the team spent three long days and three long nights rebuilding the cluster. Once it was done, they—and I—were dejected. Demoralized. Defeated. An amazing manager (who happened to be visiting our office) realized I was down, and pointed out that we'd just learned more about our new storage stack in those three days than we had in the previous three months. He reckoned a celebration was in order.

I bought some cheap sparkling wine from the local supermarket and, with another manager, took over a big conference room for a few hours. Each time someone wrote something they learned on the whiteboard, we toasted them. The team that left that room was utterly different from the one that entered it.

I'm sure Karen would have loved appreciation for her uncovering the team's weak non-code test coverage and their undocumented love of uptime-above-all-else.

Make Your Communication Clear and Your Expectations Explicit

Rather than yelling at an engineering team each time they have an outage, help them build tools to measure what an outage is, a Service Level Objective that shows how they are doing, and a culture that means they use the space between their objective and reality to choose to do the most impactful work.

When discussing failures, people need to feel safe to share all relevant information, with the understanding that they will be judged not on how they fail, but how their handling of failures improved the team, their product, and the organization as a whole. Teams with operational responsibilities need to come together and discuss outages and process failures. It's essential to approach these as fun learning opportunities, not root-cause-obsessed witch-hunts.

I've seen a team paralyzed, trying to decide whether to ship an efficiency win that would increase end-user latency by 20%. A short conversation with the product team resulted in updates to the SLO, detailing "estimated customer attrition due to different latency levels," and the impact that would have on the company's bottom line. Anyone on the team could see in seconds that low-latency was far more important than hardware costs and instead drastically over-provisioned.

If you expect someone to do something for you, ask for a specific commitment—"When might this be done?"—rather than assuming everyone agrees on its urgency. Trust can be destroyed by missed commitments.

Karen would have enjoyed a manager who told her in advance that the team considered reliability sacred and asked her to work on reliability improvements rather than optimizations.

Make Your Team Feel Safe

If you are inspired to make your team feel more psychologically safe, there are a few things you can do today:

1. Give your team a short survey (like the questions listed above), and share the results with your team.
2. Discuss what "safety" means to your team; see if they'll share when they felt "unsafe."
3. Build a culture of respect and clear communication, starting with your actions.

Treat psychological safety as a key business metric, as important as revenue, cost of sales, or uptime. This will feed into your team's effectiveness, productivity, staff retention, and any other business metric you value.

Why Are Operations Teams More Likely to Feel Unsafe than Other Engineering Teams?

We Love Interrupts and Information

Humans suck at multitasking. Trying to do multiple things at once either doubles the time the task takes or doubles the mistakes [5]. A team that's expected to make progress with project work while being expected to be available for interrupt work (tickets, on-call, walkups) is destined to fail. And yet, operations attracts people who like being distracted by novel events. Do one thing at a time. Timebox inbound communications as well as interrupt time.

Operations teams are expected to manage risk and uncertainty for their organization. We build philosophies for reasoning about risk and strategies for coping with bad outcomes, defense in depth, playbooks, incident management, escalation policies, etc. When humans are exposed to uncertainty, the resultant "information gap" results in a hunger for information, often exaggerated past the point of utility [6]. This can lead to information overload in the shape of ludicrously ornate and hard to understand dashboards, torrents of email, alerts, and automatically filed bugs. We all know engineers who have hundreds of bugs assigned to them, which they cannot possibly ever fix, but refuse to mark them "Won't Fix." Another pathology is subscribing to developer mailing lists to be aware of every change being made to the system. Our love of novelty blinds us to the lack of value in information we cannot act on.

Admit that most information is not actionable, and be brutal with your bugs, your mail filters, and your open chat apps.

On-Call and Operations

The stress of on-call is what drives people away from operations roles. Curiously, 24/7 shifts are not the problem. The real problem is small on-call rotations that result in long, frequent shifts. The more time people spend on-call, the more likely they are to suffer from depression and anxiety [7]. The expectation of having to act is more stressful than acting itself [8]. It's one thing to accept that on-call is part of a job. It's another to tell your five-year-old daughter you can't bring her to the playground.

We can mitigate this stress by ensuring on-call rotations of no fewer than six people, with time-in-lieu for those with significant expectations around response times, or personal life curtailment. Compensate based on time expecting work, not time doing work. Incident management training or frequent "Wheel of Misfortune" drills can also reduce stress, by increasing people's confidence. Ensure on-call engineers prioritize finding someone to fix a problem when multiple incidents happen concurrently [9].

Cognitive Overload

Operations teams support software written by much larger teams. I know a team of 65 SREs that supports software written by 3,500 software engineers. Teams faced with supporting software written in multiple languages, with different underlying technologies and frameworks spend a huge amount of time trying to understand the system and so have less time to improve it.

To reduce complexity, software engineers deploy more and more abstractions. Abstractions can be like quicksand. ORM (object-relational mapping) [10] is a wonderful example of a tool that can make a developer's life easy by reducing the amount of time thinking about database schemas. By obviating the need for developers to understand the underlying schema, developers no longer consider how ORM changes impact production performance. Operations now need to understand the ORM layer *and* why it impacts the database.

Monolithic designs are often easier to develop and extend than microservices. There can be valid business reasons to avoid duplication of sensitive or complex code. However, because they attract heterogeneous traffic classes and costs, they are a nightmare for operations teams to troubleshoot or capacity plan.

Everyone understands that onboarding of new, evolving software strains an operations team. We ignore the burden of mature "stable" services. There is rarely any glamorous work to be done on such services, but the team still needs to understand it. Mature services can silently swamp an operations team.

Ensure teams document the impact of cognitive load on development velocity. It has a direct and serious impact on the reliability of the software, the morale and well-being of the operations team, and the long-term success of the organization.

Imaginary Expectations

Good operations teams take pride in their work. When there is ambiguity around expectations of a service, we will err on the side of caution and do more work than needed. Do we consider all of our services to be equally important? Are there some we can drop to "best effort"? Do we really have to fix all bugs logged against our team, or can we say, "Sorry, that's not our team's focus"? Are our SLAs worded well enough that the entire team knows where their effort is best directed on any given day? Do we start our team meeting with the team's most important topics, or do we blindly follow process?

Ensure there are no magic numbers in your alerts and SLAs; if your team is being held to account for something, ensure there is a good reason that everyone understands.

Operations Teams Are Bad at Estimating Their Level of Psychological Safety

Lastly, I'll leave you with a thought: *people who are good at operations are bad at recognizing psychologically unsafe situations*. We consider occasionally stressful on-call "normal" and don't feel it getting worse until we burn out. The curiosity that allows us to be creative drives us to information overload. Despite being realistic about how terrible everything is, we stay strongly optimistic that the systems, software, and people we work with will get better.

I've given surveys to deeply troubled teams where every response seemed to indicate everything was wonderful. I'd love to hear from people who have experience uncovering such cognitive dissonance in engineers.

References

[1] J. Rozovsky, "Five Keys to a Successful Google Team": http://bit.ly/1X0Uygj.

[2] Wikipedia, "Imposter Syndrome," last edited 9/21/17: https://en.wikipedia.org/wiki/Impostor_syndrome.

[3] D. Morrison speech transcript: http://bit.ly/2fkDqnu.

[4] Intercom blog, "6 Weeks: Why It's the Goldilocks of Product Timeframes": https://blog.intercom.com/6-week-cycle-for-product-teams/.

[5] P. Atchley, "You Can't Multitask, So Stop Trying," *Harvard Business Review*, Dec 21, 2010: https://hbr.org/2010/12/you-cant-multi-task-so-stop-tr.

[6] G. Loewenstein, "The Psychology of Curiosity," *Psychological Bulletin*, vol. 116, no. 1, 1994: http://bit.ly/2xmqpOE.

[7] A.-M. Nicol and J. S. Botterill, "On-Call Work and Health: A Review," *Environ Health*, vol. 3, 2004: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC539298/.

[8] J. Dettmers, T. Vahle-Hinz, E. Bamberg, N. Friedrich, M. Keller, "Extended Work Availability and Its Relation with Start-of-Day Mood and Cortisol," *Journal of Occupational Health Psychology*, vol. 21, no. 1, Jan. 2016: https://www.ncbi.nlm.nih.gov/pubmed/26236956.

[9] D. O'Connor, "Bad Machinery: Managing Interrupts under Load," SREcon15 Europe, USENIX: http://bit.ly/2xWjbnZ.

[10] Wikipedia, "Object-Relational Mapping," last edited 7/7/17: https://en.wikipedia.org/wiki/Object-relational_mapping.

From Sysadmin to SRE in 2587 Words

VLADIMIR LEGEZA

Vladimir Legeza is a Site Reliability Engineer in the Search Operations team at Amazon Japan. For the last few decades, he has worked for various companies in a variety of sizes and business spheres such as business consulting, Web portals development, online gaming, and TV broadcasting. Since 2010, Vladimir has primarily focused on large-scale, high-performance solutions. Before Amazon, he worked on search services and platform infrastructure at Yandex.
vlegeza@amazon.co.jp

"If you cannot measure it, you cannot improve it."

—*William Thomson, Lord Kelvin*

Site Reliability Engineering is a set of practices that allow a variety of companies to run and support systems at large scale efficiently and cost-effectively. The key difference that distinguishes sysadmins from SREs is the single property: the point of observation. There is only one fundamental principle that can drive you to this relatively new field and lead you to understand all these practices, origins, adaptations, and further improvement ideas that finally will increase your users' and customers' loyalty and satisfaction.

Instead of jumping directly into the definition of principles, let's figure it out ourselves through the following examples.

Let's say a manager asked you to create a small new service: "A sort of a simplified Web crawler. It has to receive a base URL, download its content, find and return a list of all URLs retrieved from that page with the status whether it is valid and accessible or not." This task is more or less straightforward. An average software developer can immediately begin the implementation using not more than a hundred lines of high-level code.

An experienced sysadmin given the same task will, with high probability, try to understand the technical aspects of the project. For instance, she may ask questions like, "Does the project have an SLA?" and dig deeper: "What load should we expect, and what kind of outages do we need to survive?" At that point, prerequisites might be as simple as, "The load will be no more than 10 requests per second, and we expect that responses will take no longer than 10 seconds for a single URL request."

Now let's invite an SRE to the conversation. One of his first questions would be something like, "Who are our customers? And why is getting the response in 10 seconds important for them?" Despite the fact that these questions came primarily from the business perspective and did not clarify any technical details, the information they reveal may change the game dramatically. What if this service is for an "information retrieval" development team whose purpose is to address the necessity of the search engine results page's content validation, to make sure that the new index serves only live links? And what if we download a page with a million links on it?

Now we can see the conflict between the priorities in the SLA and those of the service's purposes. The SLA stated that the response time is crucial, but the service is intended to verify data, with accuracy as the most vital aspect of the service for the end user. We therefore need to adjust project requirements to meet business necessities. There are lots of ways to solve this difficulty: wait until all million links are checked, check only the first hundred links, or architect our service so that it can handle a large number of URLs in a reasonable time. The last solution is highly unlikely, and the SLA should therefore be modified to reflect real demands.

What we've just done is to raise the discussion to a new level—the business level. We started with the customer and worked backward. We understood the service's use cases, identified its key aspects, and established and adjusted the SLA. Only now can we begin to architect the solution. This is the exact meaning of the first of Amazon's leadership principles: "Customer Obsession—Leaders start with the customer and work backwards" (https://www.amazon.jobs/principles). Absolutely the same idea appears in the first of Google's "Ten Things" philosophy: "Focus on the user and all else will follow" (https://www.google.com/intl/en/about/philosophy.html).

At this point, I want to present a short, three-character terminology clarification to avoid confusion or uncertainty:

SLI: Service Level Indicator is a single, measurable metric related to service behavior that is carefully chosen with a deep understanding of its value's meaning. Every possible value should be clearly defined as "good" or "bad." Also, all barrier values that turn "good" to "bad" and vice versa should be precisely specified. It can be measured on its own terms and conditions and may have more than one axis of measurement. However, the rule of thumb is that every indicator must be meaningful.

Example SLI: 99% of all requests per one calendar year should be served in 200 ms.

SLA: Service Level Agreement is a set of SLIs that defines the overall behavior of what users should expect from the service. A good SLA represents not only a list of guarantees but also contains all possible limitations and actions that may take place in specific circumstances: for instance, graceful degradation in a case of primary datacenter outage, or what happens if a certain limit is exhausted.

SLO: Service Level Objective is absolutely the same set of SLIs that an SLA has but is much less strict and usually raises the bar of the existing SLA. The SLO is not what we have to have, but what we want to have.

Example: For an SLA, a single SLI might be defined as "99% of all requests should be served in 200 ms," and in the SLO the same indicator may look like "99.9% of all requests should be served in 200 ms."

For further details, please refer to Google's *Site Reliability Engineering* (https://landing.google.com/sre/book/chapters/service-level-objectives.html).

The principle that the user's perspective is fundamental is very powerful and leads us to the understanding of vital service aspects. Knowing what is valuable for the customer provides a precise set of expectations that have to be finally reflected in the SLA. And by being carefully crafted, the SLA may shed light on many dark corners of a project, predicting and preventing difficulties and obstacles.

But first, the SLA is designated as a reference point to understand how well a service is performing. There might be hundreds of metrics that reflect a service's state, but not all of them are appropriate for an SLA. Although the number of SLIs tends to be as minimal as possible, the final list of SLIs should cover all major user necessities.

Only two relatively simple questions should be answered positively to indicate that an investigated metric is a good candidate to be chosen, or otherwise, should definitely not be.

- Is this metric visible to the user?
- Is this metric important enough to the user (and from his/her perspective as a service customer) that it needs to be at a certain level or inside a particular range?

Internal SLAs

What if the service is not an end-customer-facing one. Should this service have its own SLA too? To clarify the "Yes" answer, let's step back a little bit from the technical terminology; we will see that the SLA itself is nothing but a list of criteria of what you can expect from the service, that is, a simple list of expectations.

When you are thinking of a new service to be used in your project, one of the first questions you want answered is, will this service meet your expectations or not? It does not matter what kind of service it is: it might be an external ticket-based authentication system, local corporate storage, or a cross datacenter distributed message-passing bus. But to figure out whether you can you use it or not, you should know what this service promises you and what limitations are applied. Hence, every producer-consumer relationship is built on certain expectations, and, hence, every service should provide a list of guarantees for all valuable expectations regardless of whether the service is an internal or external one.

To support this statement let's consider a message distribution service that consists of four main components:

- "Data receiver": accepting and registering messages
- "Data transformer": adjusting message content with data from separate external sources
- "Distributor": delivering messages to multiple endpoints
- "Consumer": receiving data from the endpoint over a "publisher–subscriber" model

At the moment, this system is working fine: no errors, no alarms.

One day, one of the top project managers comes to us and poses the following: "One of the projects we are working on now uses the 'message distribution service.' From time to time, we will need to send a huge amount of data in a short time period. Can we handle this? Or what should we have in order to use this service?"

Let's work this out gradually. Having actual numbers for the data amount is handy. Let's say it will be three times more than the maximum known peak-time value. However, this will not provide us with a clear understanding of whether we will be able to handle this traffic or not. The reason is simple: even if we know how much data is managed by the service right now during peak times, we would still need to know the break point where we reach the service's capacity limit to be able to compare it with the forecast.

Our message distribution service has several components. As we are aware, the slowest component is the one that dictates overall service capacity: the "strength of the chain is determined by the weakest link." So now we have to spend some time to establish a performance-testing environment and identify breakpoints for every component separately and determine which component is the bottleneck.

So far, we have data that will tell us about traffic-handling possibilities. And if it is fine, we are ready to go on to announce that no changes are required. Of course, businesspeople may ask another question: "Why are we over-scaled that heavily?" but that is a different story, and hopefully it is not the case.

Our calculation reveals that we can deal only with half of expected growth. And we now need to at least double throughput. The deeper we dig, the more complicated planning becomes. Even if we assume that all the components can scale linearly (and in real life, we have to prove this assumption), we are unable to compare performance directly without accounting for one more shared restriction: the delivery time.

Our goal is to pass a number of messages throughout the service from the entry point to the final consumer, and transfer it in a reasonable amount of time.

First of all, we have to provide an actual value for the "reasonable time" metric overall and for every individual component. Only then can we measure the size of an input that the application is able to receive, process, and guarantee to pass as an output inside that "reasonable time." This will lower the throughput even further. However, the positive outcome is that we can now predict the output and compare performance across components.

Time constraints are nothing but SLIs, and the maximum number of messages is the variable that has to be adjusted to process messages during spikes and not break the time limits defined in this indicator. One interesting property of an SLI is that it only rarely changes.

So far, we know:

- Where the bottleneck is. And we can predict where the bottleneck will relocate from where it is now (literally this means that now we know the next slowest component, the next after that one, and so on).
- Expectations for every component (number of messages that can be processed by a single application instance and the expected amount of time that message can spend inside this component).
- Our capacity and how much capacity is actually in use. We are also able to predict capacity drops in case of a variety of outages and make resource reservations accordingly.

And this is still not the end of the story!

External Dependencies

As you remember, the "data transformation" component has some external dependencies. The "external" means that we are only consumers and cannot control its behavior. The question is, "What capacity can these external services provide and how will their limitations affect our component's performance and scalability?" We want to know what we may expect, and, technically, we are asking for an SLA. Once we get it, we will finally have all we need to figure out what should be adjusted and where and how.

But this real-world scenario gets even more complicated. There may be many types of messages with different priorities and time restraints. It would be tough to say what we should do if the load will grow for only one data type and other types will still be definitely affected. However, by applying the "divide and conquer" principle, we declare specific criteria for every type individually; then it will be clearly noticeable what data may have experienced stagnation and how this issue should be addressed.

A short example: due to high load spikes in high-priority messages, other message deliveries will be slowed down from a few seconds to several minutes. The key question then is, "Are several minutes' delay acceptable or not?" If we know exact barrier values and can quickly identify "good" and "bad" values, then there will be no problem taking the right action. Otherwise, we will fall into a state of not knowing and can only guess what to do or whether we should do anything at all.

As you can see, SREs mostly focus on service efficiency and quality. Architecture and what stands behind is secondary, at least until a service delivers results according to a user's expectations.

Nontechnical Solutions

Technical solutions are not the limit of SLA potential, and SLAs will give you a hand in other fields as well. The SLA determines, for instance, the right time to hand a new service over to the SRE team to support something as simple as, "If a product meets expectations (i.e., does not violate an SLA), then the product is ready; otherwise, it is not."

All new software projects will have some prerequisites long before they pass architecture review and a couple of proof-of-concept models have been built. When it is believed that an application is ready to be officially launched and begin serving live production traffic, all SLIs become live, and both objectives and agreements start to count. This is the measure's starting point. Because the SLA defines statements over time (the frequently used period is one calendar year), the project should last in this state a significant portion of this time (several months or a quarter) to collect enough datapoints that confirm that the service is stable enough and that there are no agreement violation risks.

Conclusion

The SRE philosophy differs from that of the sysadmin just by the point of view. SRE philosophy was developed based on a simple, data-driven principle: look at the problem from the user and business perspectives, where "user" means "to take care of product quality," "business" equals "managing product cases and efficiency," and "data-driven" signifies "not allowing assumptions." Identify, measure, and compare all that is important. Everything else is the result of this.

If all this sounds very difficult and complicated, start with these steps:

- Divide large services into a set of smaller ones (treat each component as an individual service).
- Identify service relationships and dependencies.
- Establish an SLO for each service first and maintain it.
- Reassign SLOs to SLAs where required.

Now, as an SRE, you can control systems more accurately and can precisely know when, what, and how much you should scale up. You can do this by efficiently identifying bottlenecks and relocating them from one service to another in a controlled manner. By fine-tuning every part of system capacity to the optimal amount, you will lower costs and raise the bar for an overall positive customer experience.

Understanding Docker

KURT LIDL

Kurt Lidl is a Principal Member of the Technical Staff at Oracle, working on the Oracle Public Cloud build team. He started using BSD UNIX with 4.2 BSD, and now contributes as a Committer on the FreeBSD Project. He lives in Potomac, Maryland, with his wife and two children. lidl@freebsd.org

Editor's note: A version of this article appeared in the July/August 2017 issue of the *FreeBSD Journal*.

Docker, from Docker Inc., is a popular containerization software system for building, deploying, and running Linux applications. Docker containers offer a relatively low overhead mechanism for running multiple Linux applications where the different applications are isolated from each other. Docker offers a high-level interface to configure, build, store, and fetch Docker images. This article contains a brief review of popular virtualization technologies, an example of Docker's facilities for building containers, and a brief discussion of Docker's future evolution.

Overview of Virtualization Techniques

There are many different virtualization techniques available across the operating systems in use today. The most basic virtualization is the concept of a software process. This is the traditional virtualization that UNIX and many other operating systems have provided to different processes from early on: each user process has an independent, protected-access memory map provided through the virtual memory system of the kernel. Other more comprehensive virtualization techniques—such as software, hypervisor-based, hypervisor-based with hardware acceleration, and containerized applications—will be reviewed.

Software Virtualization/Emulation

Software-based, complete machine emulators, such as QEMU and SIMH, can emulate practically any CPU and machine architecture on the hosting machine. These types of emulators are generally fairly slow but offer complete independence between the emulated hardware and the hosting machine. The emulation software provides an instruction-by-instruction emulation of the target machine and provides a software implementation of the hardware devices of the target machine. For example, disk drives on the target are often emulated with plain files on the hosting machine. Even machines that no longer have operating hardware, such as the Honeywell DPS8M, can be emulated. In this case, the emulation is of sufficient fidelity to allow the historically significant Multics operating system to run on the emulated machine with no software changes. Another significant example of this type of emulator was the Connectix Virtual PC software, which could emulate a complete x86 computer, hosted on a PowerPC-based Mac computer. The Connectix company was purchased by Microsoft, however, and the software is no longer available.

Hypervisor-Based Virtualization

At the opposite end of the virtualization spectrum are hypervisor-based implementations. A hypervisor-based virtualization generally runs at a significant percentage of the native speed of the hosting machine. Only a small set of hypervisor-mediated system functions execute in the hypervisor, and the rest of the user code runs in the virtualized machine at native speeds. This type of virtualization is considered fairly "heavyweight" in that each virtualized machine has its own copy of whatever operating system is being run (Figure 1). One area of performance issues with this scheme is that the virtualized operating system

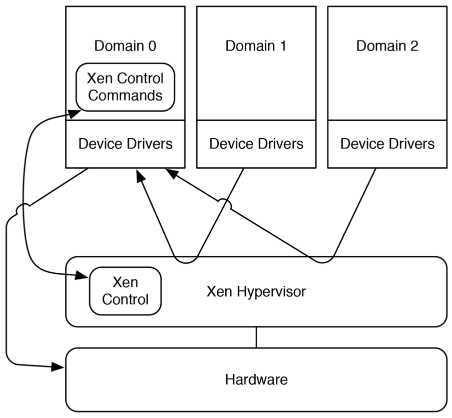


Figure 1: Xen architecture

also has to maintain its own set of memory protections for its own use. Hardware support for this type of operation, sometimes called "nested page tables," greatly enhances the operation of guest operating systems under the hypervisor.

There are many hypervisor-based virtualization platforms available, including:

- bhyve on FreeBSD
- KVM on Linux
- xhyve on Mac OS X
- Hyper-V on Microsoft Windows
- ESXi and vSphere from VMware
- Xen on multiple operating systems
- Several hardware architectures

Containerized Virtualization

Containers are lightweight virtualization schemes where processes have some sort of partitioning and isolation between different administrative groups on the same host. The different partitions all share a single kernel application binary interface (ABI) running against a single kernel instance. Often, but not always, each process running in a container can be seen on the hosting server. This type of virtualization is generally called "container computing" and offers a middle ground between the level of isolation from hypervisors and the "shared everything" from a standard UNIX environment.

Containerization is the fundamental idea behind the following facilities:

- Jail system on FreeBSD
- Control Groups on Linux
- Containers on Nexenta OS
- Containers on Solaris

Hybrid Virtualization Techniques

There are other hybrid virtualization techniques, such as running a combination of hypervisor virtual machines and then hosting various containerized applications on those virtual machines. This hybrid approach is how Docker is implemented on non-Linux machines such as the Mac OS version of Docker, which is built on top of the Mac OS xhyve virtualized machine. In a similar fashion, the Windows implementation of Docker uses the Hyper-V hypervisor to create a virtual machine running Linux, which is then used to execute the system calls from the Docker containers.

Linux Control Groups and Docker

The Linux kernel has a relatively new capability that makes Docker possible: Control Groups (aka "cgroups"). This is the fundamental technology that allows for the isolation of various user processes in one control group from affecting and directly interacting with a different control group.

In a traditional UNIX environment, there is a single hierarchy of user processes. The `init` process (pid 1) is the root of that hierarchy, and all processes can trace their ancestry back to that initial process. The cgroups facility in the Linux kernel allows for instantiating new hierarchies of processes that are contained entirely in the new hierarchy and can only interact with other processes in that hierarchy.

The cgroups facility can do more than just create new process hierarchies; it can set up resource limits (e.g., memory and network bandwidth) and attach these limits to the process hierarchies that are created. While management of the low-level cgroups mechanism via provided system utilities is possible, it is rather tedious. Docker provides a more convenient interface for controlling the cgroups mechanism at runtime, along with an easy-to-use system for building the static environments that will be executed later.

Docker uses cgroups, along with other Linux kernel facilities, such as iptables, for networking configuration and control and for a union file system (UnionFS) for isolating the container from the file systems of the hosting machine. There is also a mechanism available to allow explicit sharing of directories between the host machine and the Docker containers. The UnionFS that Docker implements is layered on top of a Docker storage driver. The storage drivers that are available depend on the particular Linux system that is running Docker and provide varying degrees of performance and stability.

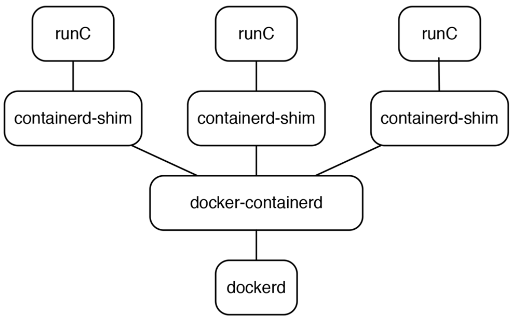


Figure 2: Docker process relationships

Docker Terminology and Software Architecture

An *image* is what Docker calls the containerized file system that has been created and loaded with the software layers that are needed for a particular application. When an image needs to run, a copy-on-write snapshot of the image is created, and that copy-on-write file system is called the *container*.

The process that starts a container is then placed into a new cgroup hierarchy. Any new processes spawned by the initial process in a Docker container will not be able to influence any processes outside of the running container, because all other processes will belong to different cgroups. This isolation prevents any interaction or interference between two or more Docker containers running on the same physical host.

A modern Docker installation typically has at least two long-running daemons, `dockerd` and `docker-containerd`. There is a single user command, `docker`, that takes multiple command keywords. This is similar to how many complex systems are controlled through a single dispatch command (e.g., `git`, `hg`, and `rndc`). The `docker` command communicates through a UNIX domain socket to the `dockerd` process. The `dockerd` process communicates with the `docker-containerd` process to specify the management of the containers on a system. There are other container software shims that are started for each container. The `runC` container runtime system initializes and starts the container, and then hands the file descriptors for `stdin/stdout/stderr` over to `containerd-shim`, which acts as a proxy of sorts between the running container and `docker-containerd` (Figure 2). This intermediate process is required so that a restart of the `dockerd` process, and therefore, the restart of `docker-containerd`, can allow the new `docker-containerd` daemon to reattach to the `containerd-shim` for each currently running container.

Docker Images Explained

A Docker image is a virtual file system, packaged as a series of layers. Each layer in the file system is stacked on top of the layers underneath it. The ultimate view of the file system is the union of the file systems that make up a Docker image. The layers in the image are built from the commands in a Dockerfile.

Dockerfile as a Recipe

The Dockerfile is a simple text file, holding one or more commands, and any comments that the user has placed in the file. When Docker builds an image, it runs each of the commands found in the file in the order they are encountered. In this manner, the `docker` build procedure is just like following a step-by-step recipe for preparing a meal. Each of the commands in the Dockerfile will generate a new layer in the resulting image. By convention, the Docker commands are written in uppercase to help differentiate them from user-specified commands. If any of the commands that are executed fail (that is, has a non-zero exit code), the building of the image stops immediately, and the image build is marked as a failure. For efficiency reasons, it is desirable to keep each layer in the Dockerfile as small as possible. This means that cleaning up after any commands that create large amounts of metadata, such as `yum update`, should be done as part of the same command that generated the metadata.

This example Dockerfile will create an image with six layers. Some of those layers, which are identical to the prior layer, will be automatically discarded during the build process.

```
An image for running Apache
FROM centos:7
MAINTAINER Ms. Nobody <nobody@example.com>
RUN yum -y --setopt=tsflags=nodocs update && \
    yum -y --setopt=tsflags=nodocs install httpd && \
    yum clean all
VOLUME ["/var/www/html", "/var/log/httpd"]
EXPOSE 80
CMD ["/usr/sbin/apachectl", "-DFOREGROUND"]
```

The first command, `FROM centos:7`, which creates the first layer in the image, specifies that the base image for CentOS 7 should be pulled from the central Docker repository into the local machine's cache of file-system layers. This layer is the bottom layer in the image. The `FROM` command must be the first command in a Dockerfile and initializes a new build.

The second command, `MAINTAINER ...`, sets a special label in the metadata for the image. This label is used to identify the creator of the image. There is also a `LABEL` command that could be used instead to set an arbitrary number of labels on an image. The labels can be used by the end user for any purpose.

The third command, `RUN yum -y update ...`, updates any out-of-date software packages that were included in the base image. The next part of the command, `yum -y install httpd`, installs the Apache `httpd` package. The final part of the command, `yum clean`, expunges all the package/repository metadata maintained by the `yum` package management system to minimize the size of the generated layer. For the same reason, the `yum` command, using the `nodocs` flag, is instructed to ignore any documentation during the upgrade and installation of packages. The arguments to the RUN command are executed by a shell process, so the complexity of the generated layer in the constructed image can be quite elaborate.

The fourth command, VOLUME [...], marks a list of directories to be used as external mountpoints in the UnionFS file system. During container execution, these mountpoints will have external file systems mounted at these locations. The UnionFS layer should not attempt to capture that activity to the copy-on-write file system. This is one method for how a container can persist data outside of the copy-on-write image from which the container is executing.

The fifth command, EXPOSE 80, provides information that will be used when a container is started from this image. A port on the hosting machine can be mapped to the specified TCP port number of the container at runtime. Or, by specifying a different networking option at runtime, the port of the container can be made accessible to other containers running on the same host.

Finally, the sixth command, CMD ..., specifies the default command to be executed when a container is started from this image. In this case, it starts the Apache Web server in the foreground. When the Apache Web server process exits, the container will be automatically stopped. It is often useful to create a small wrapper script around a daemon that is started inside a Docker container in order to restart the daemon if it stops running. By automatically restarting the daemon, the Docker container can continue to run without needing to be restarted.

Now that the purpose for each of the lines is known, building an image is straightforward. Note that some of the output from the build process has been removed and lines wrapped to improve readability. The image is created by running the command docker build *directory*, where directory is the path to the directory holding the Dockerfile.

```
docker build -t centos-apache-testimage .
Sending build context to Docker daemon  2.048kB
Step 1/6 : FROM centos:7
 ---> 3bee3060bfc8
Step 2/6 : MAINTAINER Ms. Nobody <nobody@example.com>
 ---> Using cache
 ---> 7f88dbad6a42
Step 3/6 : RUN yum -y --setopt=tsflags=nodocs update &&
           yum -y --setopt=tsflags=nodocs install httpd &&
           yum clean all
 ---> Using cache
 ---> f50595808f75
Step 4/6 : VOLUME ["/var/www/html", "/var/log/httpd"]
 ---> Running in bce2b6331fc8
 ---> 51b4c07c8eba
Removing intermediate container bce2b6331fc8
Step 5/6 : EXPOSE 80
 ---> Running in 073e6fac8709
 ---> 5bf7cadf8102
Removing intermediate container 073e6fac8709
Step 6/6 : CMD /usr/sbin/apachectl -DFOREGROUND
 ---> Running in e5a44065f0d7
 ---> 4d119d3a4776
Removing intermediate container e5a44065f0d7
Successfully built 4d119d3a4776
Successfully tagged centos-apache-testimage:latest
```

Docker Image Inspection

It is instructive to look at some of the metadata about that image, via the docker inspect command. Not all the metadata is shown in this output, just some of the more interesting pieces.

```
docker inspect centos-apache-testimage:latest
[
    {
        "Id": "sha256:db9314a42feb [...]",
        "RepoTags": [
            "centos-apache-testimage:latest"
        ],
        "ContainerConfig": {
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"/usr/sbin/apachectl\" \"-DFOREGROUND\"]"
            ],
            "Volumes": {
                "["/var/www/html",": {},
                ""/var/log/httpd"]": {}
            }
        },
        "DockerVersion": "17.06.0-ce",
        "Author": "Ms. Nobody <nobody@example.com>",
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 275797466,
        "GraphDriver": {
            "Data": null,
            "Name": "aufs"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:dc1e2dcd [...]",
                "sha256:41fc3fb9 [...]"
            ]
        }
    }
]
```

The ContainerConfig section has the complete environment specified for the processes in any containers that are started from this image. The Architecture and Os settings show that the containers support the Linux syscall interface, for the amd64 (aka x86 64) machine type. The Docker image for this article was created on a Macintosh computer, running macOS Sierra 10.12.5, but any containers will be executed with a Linux/amd64 runtime environment.

This image could be moved to any host capable of running Linux/amd64 Docker images. The portability of images is one of the principal advantages of Docker—ease of building and deploying across many different hosts without having to worry about shared library conflicts or corrupting configurations of already installed components. Docker supports the image registries where images may be stored and retrieved. A private Docker registry can be created that allows users to centrally store their customized images. Once the image is stored in the registry, a single command can retrieve the image to a host, and a second command can start a container from that image.

Running a Container

It is easy to create a running container from the example image:

```
docker run --rm -d -p 8080:80 \
    -v $(pwd)/htdocs:/var/www/html \
    -v $(pwd)/logs:/var/log/httpd \
    centos-apache-testimage:latest
```

This command starts the container, telling Docker to throw away the copy-on-write file system (--rm) when the container exits. The command runs the container in the background (-d) and port-maps localhost:8080 to the container's TCP port 80 (-p 8080:80). The command also performs volume mounts of $(pwd)/htdocs to the DocumentRoot of the Web server (-v $(pwd)/htdocs:/var/www/html), and mounts $(pwd)/logs to hold the log files from the Web server running in the container (-v $(pwd)/logs:/var/log/httpd). Finally, the name of the image to be used as the initial file system for the container is listed.

Looking at Running Containers

Once a container has been started, it will run until the initial process that started the container exits. The user who started the container, or the system administrator, can stop the container via the docker stop command. This sends a SIGTERM to the initial process in the container, and then a SIGKILL after a grace period, if the container has not stopped. This is very similar in intent to running the shutdown command on a UNIX host. The docker kill command just sends a SIGKILL to the root process in the container.

The docker ps command gives the list of running containers. The docker rm command can be used to remove the copy-on-write file system of a stopped container.

Looking at Images on the Machine

The docker images command will show the list of images currently available on the host.

The docker rmi command can be used to remove a reference to an image, freeing the storage associated with that image when the reference count drops to zero. Note that shared layers in the image may also be used by other images, so there often isn't a one-to-one correspondence between the amount of space listed as in use for the image and the amount of disk space used by the image. Many of the issues with double-counting of storage blocks that occur with file system snapshots are also evident with the docker-containerd storage of images.

Other Docker Commands

There are other Docker commands available that can be used for launching containers automatically and maintaining a set of containers that must run together to accomplish a given task. It is beyond the scope of this article to fully example the complete set of commands available inside of Docker. More advanced orchestration of multiple containers running across multiple hosts is possible via the Docker Swarm support in recent versions of Docker.

Comparison with FreeBSD Jails

Docker containers are similar to FreeBSD jails in terms of what virtualization is provided and how machine resources are shared. Both offer compartmentalized processes running against a single kernel image on the hosting machine (Figure 3). Docker offers an easy-to-use command line interface for creating, deploying, running, and updating images. Little setup and configuration are required on the hosting machine, other than the basic Docker Engine installation. FreeBSD jails have a considerably simpler interface to running and stopping jails. The base FreeBSD system offers essentially no high-level support for the building and installation of jails into a directory. There are several add-on FreeBSD ports (e.g., ezjail, qjail, and qjail4) that attempt to make jail usage less cumbersome, to varying degrees of success.

One significant advantage that Docker has over jails is the concept of spawning a per-instance copy-on-write file system for each container that is started. This is fundamental to the deployment and reusability of Docker images, whereas each FreeBSD jail typically runs in a persistent file system tree. Some of the add-on jail management systems use ZFS's snapshot and promote features to create a clone of a prototype file system

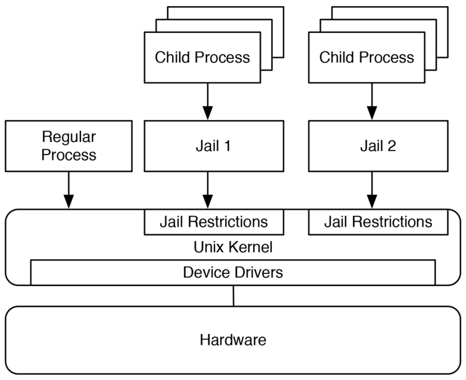


Figure 3: Jail architecture

for a newly instantiated jail, but that clone still persists the file hierarchy across multiple restarts of the jail.

With Docker containers, the Docker infrastructure takes care of mounting various directories when the container is started, whereas mounting of any directories, even including the crucial `devfs /dev mount`, must be handled explicitly for each FreeBSD jail.

For instance, running the example Docker container for Apache, there are several mountpoints active:

```
df
File system 1K-blocks       Used  Available  Use%  Mounted on
none         61890340     863500   57859916    2%  /
tmpfs         1023384          0    1023384    0%  /dev
tmpfs         1023384          0    1023384    0%  /sys/fs/cgroup
/dev/sda2    61890340     863500   57859916    2%  /etc/hosts
shm             65536          0      65536    0%  /dev/shm
osxfs       976426656  624064128  352106528   64%  /var/www/html
tmpfs         1023384          0    1023384    0%  /sys/firmware
```

The Docker system manufactured the required mounts automatically, with the exception of the `/var/www/html` mount, which was specified on the command line when the container was started.

The security benefits of Docker vs. jails are roughly equivalent. The security features of Docker are typically modified through command-line flags, while the security features for jails are either globally specified via sysctl settings or have per-jail configuration settings in the /etc/jail.conf file. Jails, when operated with the VIMAGE networking option, have a per-instance network stack. This implies that each jail could have different packet filtering in place. All the running containers on a Linux-based host share a single iptables-based packet filter configuration.

The control aspects of Docker vs. jails are quite different. Docker has many commands and options to allow almost all configurations to happen on the command line. FreeBSD jails rely heavily on the contents of the /etc/jail.conf file to specify which jails are to be run and how they are to be configured. Docker internalizes much of the configuration that is the metadata for a given Docker image. By attaching this metadata to the image, the deployment to a new host is significantly eased. FreeBSD jails have no such metadata directly attached to each jail.

In a related area, some of the FreeBSD ports for helping to manage bhyve virtualized host instances, such as iohyve, offer some of the same type of configuration help. These systems use ZFS properties to attach the metadata about a virtualized machine to a ZFS file system or ZFS zvol, which represents the file system for the virtualized machine. Several of the earliest versions of these management tools just used well-known names for the parameters that were to be controlled: hostname, number of CPUs, amount of memory, and so forth. None offered a generic, extensible tag:value configuration file that could be attached to a ZFS file system, although this might have changed since the earliest attempts at supporting virtual machine metadata in this manner.

Future Directions for Docker

The Docker system is undergoing fairly rapid evolution. There is a consortium of companies that have formed the Open Container Initiative (OCI). Significant members of the OCI include Amazon, AT&T, Cisco, Docker Inc., Facebook, Google, IBM, Microsoft, Oracle, Red Hat, and VMware. OCI is attempting to standardize both a container runtime system ("runtime-spec") and an image specification ("image-spec"). As a starting point for the standardization process, Docker Inc. donated their container specification, and their runtime system (runC) to OCI. Some of the members of OCI have vested interests in supporting more than just a Linux syscall ABI container, and the specifications are clear in the need to support multiple ABIs, as well as multiple operating systems hosting the container runtime. A recent development in the standardization process is Oracle's release of an open-source implementation of the `oci-runtime` called Railcar, which is written in Rust.

Docker on FreeBSD

Examining the current state of the Docker system, it seems that there are no insurmountable technical impediments to making the Docker system run natively on FreeBSD. The future of Docker and the support for different ABIs across containers implies that supporting a native FreeBSD kernel ABI for the containers would be possible. Obviously, this makes deployment using Docker less of a Linux/amd64 monoculture. Currently, Docker is effectively only running the Linux ABI on amd64 hardware. The Docker community, through the OCI, has already tentatively agreed to a multi-architecture system where both Linux and Windows will be supported as first-class ABI environments across multiple hardware platforms. This cross-system support is already available in a limited fashion for the Linux ABI on IBM's Z-System hardware, and nascent support for the arm64 architecture is available as well. It should be possible to extend this multi-ABI future to include FreeBSD.

Save the Date!

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

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

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

The Call for Papers is now available.
Abstract registrations are due April 26, 2018.

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

www.usenix.org/osdi18

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Save the Date!

2018 USENIX Annual Technical Conference

JULY 11–13, 2018 • BOSTON, MA, USA

USENIX ATC '18 will 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 and network management and troubleshooting, cloud and edge computing, security, privacy, and trust, mobile and wireless, and more.

The Call for Papers is now available.
Paper submissions are due February 6, 2018.

Program Co-Chairs:
Haryadi Gunawi, *University of Chicago,* and Benjamin Reed, *Facebook*

Co-located with USENIX ATC '18

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

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

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

www.usenix.org/atc18

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

COLUMNS

raise SystemExit(0)

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

It's with some regret that this is the final installment of my regular Python column. I suppose that I should offer some final words of wisdom, a historical retrospective, or maybe even a forward-looking "Python of the future" vision—however, I'm simply not that clever. Truth be told, I'm simply stretched a bit thin these days and think it would be a good time to step aside to make room for a fresh new voice and insights. So, in the spirit of leaving on a useful note, I thought I'd end on a practical trifle of a matter of some importance—the problem of getting a Python program to quit.

Bailing Out

Suppose your program has reached the final limit of what it can tolerate and you want it to die. To make it happen, raise a `SystemExit` exception and be done with it. For example:

```
raise SystemExit(1)
```

It is standard practice to include some kind of numeric exit code, which indicates success (zero) or failure (non-zero) back to the process that launched Python. Alternatively, you can include a diagnostic message.

```
raise SystemExit('Goodbye cruel world')
```

When you give a message, it is printed to `sys.stderr` and Python exits with a status code of 1. Problem solved—your program gracefully cleans up after itself and quits. Of course, that's naturally not the end of the story or else this would be a pretty short article. Let's continue.

Catching Exceptions

The `SystemExit` exception is not grouped with other exceptions. For example, it's somewhat common to encounter code that catches all errors like this:

```
try:
    something_complicated()
except Exception as e:
    print("It didn't work. Reason: %s", e)
```

This code will catch most errors, but not `SystemExit`. If you wanted to catch that, you'd have to add an extra clause for it. For example,

```
try:
    something_complicated()
except Exception as e:
    print("It didn't work. Reason: %s", e)
except SystemExit as e:
    print("I see you're going away")
    raise
```

In practice, it's rarely the case that you would ever catch `SystemExit` yourself. Even if you did, the most sensible action is perhaps to simply log the event and re-raise the exception as shown.

A common confusion is that sometimes programmers catch SystemExit by accident by writing sloppy exception handling code:

```
try:
    something_complicated()
except:
    print("It didn't work.")
```

This actually catches all possible errors, including SystemExit. However, this behavior is often unexpected and a potential source of obscure bugs. It's usually best to catch just the exceptions you need as opposed to casting such a wide net.

Keeping the Interpreter Alive

For the purpose of debugging, sometimes it's nice to keep the interpreter alive so that you can go poking around. Use python -i for that. It works even in programs that intentionally raise a SystemExit exception. You might see a message printed, but otherwise, you'll be dropped into the interactive Python shell afterwards. From there, you can mess around. For example:

```
% python3 -i spam.py
Traceback (most recent call last):
  File "spam.py", line 5, in
    spam()
  File "spam.py", line 3, in spam
    raise SystemExit("I'm dead")
SystemExit: I'm dead
>>>
```

Cleanup Actions

Sometimes you might want extra actions to take place upon program termination. For this, you can use the atexit module [1]. For example:

```
import atexit

def goodbye():
    print("So long and thanks for all of the fish")

atexit.register(goodbye)
```

atexit allows you to register an arbitrary number of zero-argument functions that get fired upon termination of the Python interpreter. The functions execute in reverse order of registration. If you need to carry extra information, it is standard practice to use a lambda or functools.partial to do it. For example:

```
def spam(name):
    atexit.register(lambda: print('Goodbye', name))
    ...
```

If needed, you can also unregister a previously registered function using atexit.unregister().

Cleanup with Context Managers

On the subject of cleanup, when working with objects, it's usually best to make use of context managers and Python's with statement. For example, suppose you had some object that involved closing a resource such as a file or connection. A good way to clean it up is to give it __enter__() and __exit__() methods like this:

```
class Spam(object):
    def __init__(self):
        self.resource = SomeResource()
        ...
    def __enter__(self):
        return self
    def __exit__(self, *args):
         # Cleanup
        self.resource.close()
```

With this object, you can now write code like this:

```
with Spam() as s:
    ...
     # Use s
    ...
 # Resources released here
```

When control-flow leaves the indented block, the __exit__() method will run. This happens regardless of what happens in the block—including SystemExit.

The __del__ Puzzle

Sometimes user-defined classes will define a __del__() method for the purposes of cleanup. For example:

```
 # spam.py
import datetime

class Spam(object):
    def __del__(self):
       print("%s destroyed at %s" % (self, datetime.datetime.now()))
```

__del__() is a particularly troublesome method to be defining in general. The main problem is that you simply don't know when it's actually going to fire. This is especially true on program exit. When Python shuts down, all active objects get garbage collected—this includes functions, classes, and modules. In the above example, it's entirely possible you could get a warning message like this printed to standard error:

```
Exception AttributeError: "'NoneType' object has no attribute
'datetime'" in <bound method Spam.__del__ of <spam.Spam
object at 0x1007d9f90>> ignored
```

What's happened here is that the datetime module reference has already been garbage-collected and is no longer defined in global scope. The __del__() method blows up because datetime is gone. This sort of thing can often be fixed by playing weird games with default arguments. For example:

```
 # spam.py
import datetime

class Spam(object):
    def __del__(self, now=datetime.datetime.now):
        print("%s destroyed at %s" % (self, now()))
```

Ugh. Explaining something like that to your coworkers is going to be hard and even then, it's no guarantee that it's going to work (what if the `now()` function itself needs other functionality that's already been garbage-collected?). The bottom line is don't rely on `__del__()` to perform cleanup actions properly when it comes to program exit. You're better off considering a context manager or a more explicit approach.

Threads

Program exit becomes much more interesting when you start programming with threads [2]. Consider the following code:

```
import threading
import time

def countdown(n):
    while n > 0:
        print('T-minus', n)
        time.sleep(5)
        n -= 1

threading.Thread(target=countdown, args=(5,)).start()
raise SystemExit("Goodbye")
```

If you run this, program termination is delayed until the thread runs to completion. In fact, if you had a lot of threads, program exit won't occur until all of them terminate.

This situation is made even more unfortunate given that there is no mechanism for terminating or signaling a thread once started. Your only sane recourse is to build in some kind of periodic polling or check.

```
import threading
import time

main_thread = threading.current_thread()

def countdown(n):
    while n > 0 and main_thread.is_alive():
        print('T-minus', n)
        time.sleep(5)
        n -= 1

threading.Thread(target=countdown, args=(5,)).start()
raise SystemExit('Goodbye cruel world')
```

Alternatively, you could create the thread as "daemonic" like this:

```
import threading
import time

def countdown(n):
    while n > 0:
        print('T-minus', n)
        time.sleep(5)
        n -= 1

threading.Thread(target=countdown, args=(5,), daemon=True).
start()
raise SystemExit("Goodbye")
```

Daemonic threads are killed immediately once the main thread exits. This is not without its own set of concerns, however. Daemonic threads killed in this way do not exit gracefully. For example, they don't garbage-collect remaining objects (they don't execute `__del__()` methods), and they don't run the `__exit__()` method of context managers. So if you were expecting some kind of graceful cleanup from this, don't.

Curiously, functions registered with `atexit` will still run upon termination of a threaded program—even if registered by daemonic threads. So you could write code like this:

```
import threading
import time
import atexit

def countdown(n):
    onexit = lambda: print('Thread dead. Final value', n)
    atexit.register(onexit)
    while n > 0:
        print('T-minus', n)
        time.sleep(5)
        n -= 1
    atexit.unregister(onexit)

threading.Thread(target=countdown, args=(5,), daemon=True).
start()
time.sleep(12)
raise SystemExit('Goodbye cruel world')
```

When you run this, you'll see a message about a final value of 3.

Keyboard Interrupts and Signals

One especially nasty problem with program termination is the handling of keyboard interrupts (Control-C) and signals [3]. These events often ultimately result in program termination, but unlike a typical `SystemExit` exception, they occur asynchronously. This means that they could potentially occur on any statement in your program.

Perhaps the most important thing to note about signals is that they are only handled by Python's main execution thread. There are situations where it is impossible to receive signals and it will appear as if it is impossible to kill your program. The most common scenario is if the main program gets tied up on a lock or becomes busy with some CPU-intensive task. Here is a simple example you can try:

```
>>> 'a' in range(1000000000)    # Use xrange on Python2
<Ctrl-C>
```

In this example, you'll find that the code becomes totally unresponsive to the keyboard interrupt until the operation completes. Under the covers the interpreter is tied up with a big calculation taking place in C. There's just no opportunity for it to be interrupted.

More diabolical situations can arise with combinations of locking and signal handling. Consider this interesting bit of code involving the logging module:

```
import logging
import time
import signal

log = logging.getLogger(__name__)

def goodbye(signo, frame):
    log.debug('Goodbye')
    raise SystemExit()

def spin():
    while True:
        log.debug('Hey %f' % time.time())

signal.signal(signal.SIGINT, goodbye)
logging.basicConfig(level=logging.DEBUG)
spin()
```

In this code, a constant stream of log messages is quickly emitted until terminated by a Control-C (SIGINT). It might look innocent enough and it might even seem to work when you try it. However, there are hidden dangers. It turns out that the logging module internally uses thread locks. If you run this program repeatedly, killing it with Control-C, you might find that just every so often, instead of dying, the whole program freezes. What happened? The main program was in the middle of logging a message with the lock held when a signal arrived. The signal handler then tried to log a message, but is now deadlocked due to the logging lock being in use. Your only recourse here—open up another terminal and kill Python using `kill -9`.

Some general advice concerning threads, signals, and program exit. If you want your program to terminate, a sensible strategy is often one that keeps the main-thread free for nothing other than signal handling. Use it to catch keyboard interrupts and other signals and have it arrange to have the rest of the program exit in the most graceful manner that you can devise. This is only a simple template:

```
import threading
import time

terminated = False

def countdown(n):
    while n > 0 and not terminated:
        print('T-minus', n)
        time.sleep(5)
        n -= 1

threading.Thread(target=countdown, args=(5,)).start()

 # Main-thread. Spin and wait for termination
try:
    while True:
       time.sleep(1)
finally:
    terminated = True
```

There are many variations on this theme, but if you've got a very complicated application and it involves concurrency, directing all asynchronous signals and/or the keyboard interrupt to a single well-defined place is probably a good strategy.

The Nuclear Option

Finally, if all else fails, there is always `os._exit()`. For example:

```
import os; os._exit(1)
```

This is a direct line to the underlying `exit()` system call. It will terminate Python immediately, with no cleanup of any kind. As a general rule, though, you'd probably want to avoid this except as a last resort.

Final Words

As noted, this is my last installment of the regular Python column. I'd just like to thank Rik Farrow and everyone else at USENIX for their support over the last six years and hope that you've enjoyed it. I intend to stay active in the Python community, so say hello if you ever see me at a conference, or if you happen to be in the Chicago area, please feel free to look me up. Until then, happy Python hacking!

References

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

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

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

Practical Perl Tools

Perl without Perl

DAVID N. BLANK-EDELMAN

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

It's not exactly a Zen koan, but it will have to do for this issue's column. Today we're going to talk about how a method for parsing Perl without using the actual Perl interpreter can offer a whole host of benefits. This idea may be somewhat surprising because Perl has a reputation (perhaps deserved) as being a language where "the only thing which can parse Perl (the language) is perl (the binary)." For some detailed examples of this criticism, see one of the well-known essays at http://www.perlmonks.org/index.pl?node_id=44722, which is where that quote came from, originally attributed to Tom Christiansen.

Part of the problem is that there are some ambiguities in the language that only resolve themselves definitively during runtime. This reality harshed many the mellow of aspiring tool creators until at some point someone asked the question, "Could we write something that could parse enough Perl to be useful? Maybe we won't get 100% correct behavior, but how good does it have to be to let us get real work done?" Turns out, we can actually get really, really close. The leap in thinking that made this possible was a small shift in mindset: instead of thinking of the program/file as code that is executed, we can think of it as a static document we can parse. This lets us get close enough that very useful tools can be created, and that's what this column will focus on.

PPI

The heart of all of this work is the PPI module and the small ecosystem that surrounds it. PPI is a Perl module (so perhaps the title isn't entirely accurate) that knows how to parse existing Perl code. Even though this module is at the center of everything we're going to talk about today, we're going to do almost nothing with it directly. Directly using PPI is easy (excerpted from the docs):

```
use PPI;

# Create a new empty document
my $Document = PPI::Document->new;

# Load a Document from a file
$Document = PPI::Document->new('Module.pm');

# Does it contain any POD?
if ( $Document->find_any('PPI::Token::Pod') ) {
   print "Module contains POD\n";
}

# Remove all that nasty documentation
$Document->prune('PPI::Token::Pod');
$Document->prune('PPI::Token::Comment');
```

```
        # Save the file
        $Document->save('Module.pm.stripped');
```

Basically, you create the object that will represent the Perl document and then tell PPI to parse the file (or a chunk of Perl code in a string, not shown here). In the code above we tell PPI to find or remove different Perl structures from the parsed info and write this document back out to disk. This is pretty simple, so what can we build on this model?

Make It Pretty

One thing we can do once we "understand" the Perl document that has been parsed is to output the document in a way that improves its readability. One example of this is `PPI::Prettify`.

`PPI::Prettify` bootstraps on the work done by Google to produce a syntax highlighting tool that would make it easy to embed readable code in a Web page. This is the same software that Stack Overflow uses for its code examples. Their package, *prettify.js* (https://code.google.com/archive/p/google-code-prettify/), consists of a JavaScript module to do the highlighting and accompanying CSS that lets you "theme" the results. `PPI::Prettify` lets you skip the JavaScript part and just make use of the CSS themes that work with prettify.js (plus the highlighting is allegedly more accurate).

Using the module is basically a single call:

```
        use File::Slurp;
        use PPI::Prettify 'prettify';

        my $document = read_file( $ARGV[0] );

        print prettify( { code => $document } );
```

Here you see me using `File::Slurp` to pull an entire file into memory because `PPI::Prettify` expects to have code fed to it via a scalar.

The output looks a little like this:

```
        <pre class="prettyprint"><span class="kwd">use</span
   ><span class="pln"> </span><span class="pln">WebService
   ::Spotify</span><span class="pun">;</span><span class="pln">
        </span><span class="pln">
        ...
```

which is only "pretty" when displayed in an HTML document that references the right prettify.js CSS file for those classes.

Count It

Another helpful class of things built on top of PPI are the modules that can give us some stats about our code. For example, the Perl::Metrics::Simple package comes with a *countperl* utility, which gives the following output:

```
$ countperl spotify2.pl
Perl files found: 1

Counts
--------
total code lines:           14
lines of non-sub code:      8
packages found:             0
subs/methods:               1

Subroutine/Method Size
----------------------------
min:                        6
max:                        6
mean:                       6.00
std. deviation:             0.00
median:                     6.00

McCabe Complexity
---------------------
Code not in any subroutine
min:                        5
max:                        5
mean:                       5.00
std. deviation:             0.00
median:                     5.00

Subroutines/Methods
min:                        2
max:                        2
mean:                       2.00
std. deviation:             0.00
median:                     2.00

List of subroutines, with most complex at top
-----------------------------------------------------
complexity sub                                path         size
  5        {code not in named subroutines} ./spotify2.pl    8
  2        print_artists                   ./spotify2.pl    6
```

If you really get into this sort of static analysis, there are more complex modules like `Perl::Metrics` and `Code::Statistics` you may want to explore.

Find It

A perhaps more exciting consequence of being able to parse Perl from Perl is the ability to create utilities that can selectively operate on source files based on the semantics of the code they contain. For example, we can now write programs that only work on Perl files that contain documentation (or better yet, are missing documentation). We can search for text just in the comments of the code (looking at only real comments vs. a crude guess that looks at strings that start with a #). We can look for code that has "barewords" in it, and so on. PPI opens this all up for us.

Two easy ways to get into this are using the Find::File::Rule:PPI module and utilities like App::Grepl. Let's look at both.

We've talked about the Find::File::Rule family before in this column (I'm very fond of it), but let's do a quick review anyway. Find::File::Rule is a module family meant to extend the functionality of the Find::File module that ships with Perl and also make it easier to use. Instead of writing a special subroutine whose job it is to determine whether an object found when traversing a directory tree is of interest, you write code that looks more like this (from the doc):

```
# find all the subdirectories of a given directory
my @subdirs = File::Find::Rule->directory->in( $directory );
```

and

```
# find all the .pm files in @INC
my @files = File::Find::Rule->file()
                            ->name( '*.pm' )
                            ->in( @INC );
```

Basically, you string together a bunch of methods that express rules for determining the files or directory names of interest. I find it easiest to read the code backwards—in the last code sample, it says to look in the directories listed in the @INC array. In those directories, collect the names of all of the files and directories that have a name ending in .pm. Of these, return those that are files.

Find::File::Rule:PPI adds a ppi_find any method that lets you specify the same sort of selectors we saw at the very beginning of the column. So, for instance, if we wanted a list of all of the Perl files in a directory (and its subdirectories) that have embedded POD documentation, we could write:

```
use File::Find::Rule;
use File::Find::Rule::PPI;

my @podfiles =
  File::Find::Rule
   -> file()
   -> name('*.pm')
   -> ppi_find_any('PPI::Token::Pod')
   -> in('.');

print join("\n", @podfiles), "\n";
```

App::Grepl takes this a little further in that you can search for text (à la grep) in specific Perl structures. For example, you could look for the string "USENIX" in just the POD part of files with code like this:

```
use App::Grepl;

my $grepl = App::Grepl->new( {
    dir      => ".",
    look_for => [ 'pod' ],
    pattern  => 'USENIX'
} );
$grepl->search;
```

or from the command line:

```
        grepl --dir . --pattern 'USENIX' --search pod
```

There are modules that can do less general scanning as well. For example, App::ScanPrereqs offers a nice CLI that can show all of the prerequisites for the code in a directory:

```
$ scan-prereqs .
+-----------------------------------------+-------------+
| module                                  | version     |
+-----------------------------------------+-------------+
| blib                                    | 1.01        |
| ExtUtils::MakeMaker                     | 0           |
| File::Find                              | 0           |
| File::Spec                              | 0           |
| Filename::Backup                        | 0           |
| IO::Handle                              | 0           |
| IPC::Open3                              | 0           |
| Log::ger                                | 0           |
| Module::CoreList                        | 0           |
| Perinci::CmdLine::Any                   | 0           |
| perl                                    | 5.010001    |
| Perl::PrereqScanner                     | 0           |
| Perl::PrereqScanner::Lite               | 0           |
| Perl::PrereqScanner::NotQuiteLite       | 0           |
| Pod::Coverage::TrustPod                 | 0           |
| strict                                  | 0           |
| Test::More                              | 0           |
| Test::Pod                               | 1.41        |
| Test::Pod::Coverage                     | 1.08        |
| warnings                                | 0           |
+-----------------------------------------+-------------+
```

It's a little meta, but that's what the module reports for itself when I run the scanner.

A similar tool comes from the Perl::MinimumVersion module which can output information like:

```
 -----------------------------------------------------------------------
| file                            | explicit  | syntax   | external |
| --------------------------------------------------------------------- |
| Makefile.PL                     | v5.10.1   | v5.10.0  | n/a      |
| bin/scan-prereqs                | v5.101    | v5.6.0   | n/a      |
| lib/App/ScanPrereqs.pm          | v5.10.1   | v5.10.0  | n/a      |
| t/00-compile.t                  | v5.6.0    | v5.6.0   | n/a      |
| t/author-pod-coverage.t         | ~         | ~        | n/a      |
| t/author-pod-syntax.t           | ~         | v5.6.0   | n/a      |
| t/release-rinci.t               | ~         | ~        | n/a      |
| --------------------------------------------------------------------- |
| Minimum explicit version    : v5.10.1                                 |
| Minimum syntax version      : v5.10.0                                 |
| Minimum version of perl     : v5.10.1                                 |
 -----------------------------------------------------------------------
```

If you want to go one step fancier, there's the `App::PrereqGrapher` module which makes pretty pictures like the one in Figure 1.

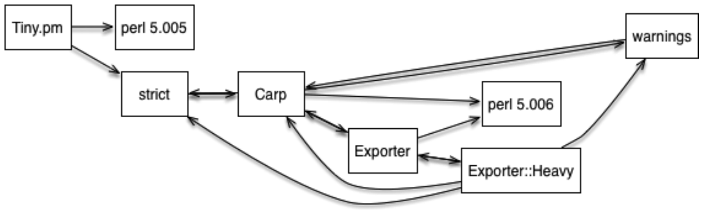


Figure 1: Output from App::PrereqGrapher

Make Your Code Better

Okay, the last set of PPI-powered modules to cover: those that help us write better code. Examples of this are the several modules like `Log::Report::Extract::PerlPPI` that make it easier to find, extract, and replace translatable strings in the code (e.g., error messages) for when you need to write code that will work in several languages.

Even more fun is software like `Code::DRY`, which calls itself "Cut-and-Paste-Detector for Perl code" and says, "The module's main purpose is to report repeated text fragments (typically Perl code) that could be considered for isolation and/or abstraction in order to reduce multiple copies of the same code (aka cut and paste code)." This can produce reports like (from the doc):

```
1 duplicate(s) found with a length of 8 (>= 2 lines) and 78 bytes
    reduced to complete lines:
1.  File: t/00_lowlevel.t in lines 57..64 (offsets 1467..1544)
2.  File: t/00_lowlevel.t in lines 74..81 (offsets 1865..1942)
===================
...<duplicated content>
===================
```

As a last and perhaps most useful module to visit, we return to something we've seen in past columns: `Perl::Critic`. `Perl::Critic` attempts to "critique Perl source code for best-practices." I think its doc says it best:

> `Perl::Critic` is an extensible framework for creating and applying coding standards to Perl source code. Essentially, it is a static source code analysis engine. `Perl::Critic` is distributed with a number of `Perl::Critic::Policy` modules that attempt to enforce various coding guidelines. Most Policy modules are based on Damian Conway's book Perl Best Practices. However, `Perl::Critic` is not limited to PBP and will even support Policies that contradict Conway. You can enable, disable, and customize those Policies through the `Perl::Critic` interface. You can also create new Policy modules that suit your own tastes.

When I write code, I tend to use a few tools to improve it, even as I'm writing. First, there's the internal checks of use strict. Then there is perltidy (which doesn't use PPI because it existed a couple of years before PPI came into being, but there are bug reports that suggest it should) for aligning and generally pretty printing the code. And finally, there's perlcritic, the command line tool that calls `Perl::Critic` on the code to look for best practices being violated by the code. For example, here's a run on the `CSS::Tiny` module file:

```
$ perlcritic Tiny.pm
Bareword file handle opened at line 27, column 2.  See pages
  202,204 of PBP.  (Severity: 5)
Don't modify $_ in list functions at line 53, column 16.  See
  page 114 of PBP.  (Severity: 5)
Expression form of "eval" at line 69, column 19.  See page
  161 of PBP.  (Severity: 5)
Expression form of "eval" at line 69, column 46.  See page
  161 of PBP.  (Severity: 5)
Bareword file handle opened at line 90, column 2.  See pages
  202,204 of PBP.  (Severity: 5)
```

Not everything it complains about is crucial to change, but it does occasionally point out flaws in the code that can and should be easily remedied.

With this tip, I'll say take care, and I'll see you next time.

Go
HashiCorp's Vault

CHRIS "MAC" MCENIRY

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

In the Fall 2017 issue, we examined using Go to set up TLS encryption between our gls service and gls client. To recap, Go has three strong libraries which we used: the crypto/x509 library and the crypto/rsa libraries provided us with a means of generating the certificate, and the crypto/tls library provided us with a way of wrapping the gls communications with encryption.

In this issue, we're going to look to another tool to handle our certificate and key generation: HashiCorp's Vault (https://www.vaultproject.io/).

Vault, which is written in Go, has a Go client library for it. Not surprisingly, this library is not in the standard Go library. This will give us a chance to get a taste of the new Golang dependency management tool: dep (https://github.com/golang/dep).

We're going to use the existing code from last issue's article, but we've added a new file: `certs/generate_certs_vault.go` to do our certificate generation. You can get this code from https://github.com/cmceniry/login-glss, or by running:

```
shell$ go get -u github.com/cmceniry/login-glss
```

Using an External Secret Store

Organizations are under increasing pressure from regulatory entities to ensure proper handling of secrets. They need to be able to demonstrate a proper chain of custody and limited exposure of those secrets. They need to be able to show when a secret was accessed and by whom.

This ends up involving a significant amount of overhead and having a large impact on code and configuration processes. The secret cannot be kept with other configuration data, even though it is critical to the application or service being able to run.

What are some of these secrets? A few examples are:

- Passwords for service accounts to access databases
- Salts or shared secrets for message hashing and signatures
- Shared secrets for encryption channels
- TLS keys or the passphrases to TLS keys

Imagine having to go to every location where an application is running and manually putting a password or passphrase in place. The application code and the rest of its configuration are already there, but you still can't start the application without having these secrets. Even in a well-run organization, this can cause critical delays to service delivery or restoration. And that this is not well auditable is just as bad. You have to rely on people filling out sign in/sign out forms for retrieving the password or passphrase.

Vault makes it easier and faster to handle the secret and to keep that handling auditable. It is a network service that can share or issue secrets. The application or application server authenticates itself to Vault and receives an access token in response. The application then uses this token to retrieve any secrets it needs. Vault is maintaining the audit log for when that secret was created, modified, or retrieved.

Vault is designed to have multiple pluggable back ends. A back end handles a particular type of secret—such as a generic or database password. In the case of the database password, Vault can perform an action to create the credentials on the fly when it is asked for the password. This allows limited-use passwords and other precautions to reduce risk.

We're going to use Vault to generate our keys and certificates using the Vault Go library. But before we do that, we need to set up Vault and prepare it to be a certificate authority.

Getting Started: Vault

To get set up with Vault for this exercise, we're going to:

- Install Vault
- Start the Vault server
- Set our authentication credentials for Vault
- Configure Vault with our certificate authority back end
- Have Vault generate a key and certificate for our certificate authority
- Configure a role to use our certificate authority

Vault is a combined network server and client in one binary. You can download the binary for several platforms from the project's Web site: https://www.vaultproject.io/downloads.html.

To keep this article brief, we're going to cut a few corners when starting the server—namely, start it in `dev` mode. This leaves out the certificates for encrypting the communication with `vault`, and shortcuts the authentication phase by using a predefined token available in `dev` mode. Vault will keep everything in memory so this is definitely not a permanent installation. In a production deployment, you would want to examine both of these areas more closely.

Start `vault` with the `-dev` and `-dev-root-token-id=mytoken` and send it to the background.

```
shell$ ./vault server -dev -dev-root-token-id=mytoken &
[1] 13625
shell$ ==> Vault server configuration:
                     Cgo: disabled
         Cluster Address: https://127.0.0.1:8201
              Listener 1: tcp (addr: "127.0.0.1:8200", cluster address:
"127.0.0.1:8201", tls: "disabled")
               Log Level: info
        Mlock: supported: false, enabled: false
        Redirect Address: http://127.0.0.1:8200
                 Storage: inmem
                 Version: Vault v0.8.3
             Version Sha: 6b29fb2b7f70ed538ee2b3c057335d706b6d4e36

==> WARNING: Dev mode is enabled!
```

Next, we'll want to set up a few environment variables. `VAULT_ADDR` sets the connection point for Vault. `VAULT_TOKEN` sets the authentication token to use.

```
shell$ export VAULT_ADDR=http://127.0.0.1:8200
shell$ export VAULT_TOKEN=mytoken
```

Now we can set up the back end in Vault. In this case, we're going to use the `pki` back end, which will be our certificate authority. We want to make it available inside of Vault at a known location—we'll use `myca`. Back ends are set up with the `mount` command.

```
    shell$ ./vault mount -path=myca pki
    2017/09/23 21:22:55.762379 [INFO ] core: successful mount:
  path=myca/ type=pki
    Successfully mounted 'pki' at 'myca'!
```

Now we need to have Vault generate the key and certificate for our certificate authority. Vault uses a generic interface to the back ends—namely, you can perform write (Create/Update), read (Read), and delete (Delete) operations on paths inside of Vault. When a back end is mounted, it exposes child paths underneath the mount path. You can perform CRUD operations on these child paths as appropriate for the back end. The paths and their usages for the `pki` back end can be found at https://www.Vaultproject.io/api/secret/pki/index.html. We're going to start by issuing a write to the path for "Generate Root." For this path, we have to specify the common name that will be stamped on the CA's certificate.

```
    shell$ ./vault write myca/root/generate/internal common_
  name="My CA"
    Key          Value
   _        -----
    certificate    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    expiration     1508992653
    issuing_ca     -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    serial_number   54:64:79:74:5c:b8:a1:6a:66:0c:88:6e:eb:bb:
  40:1e:46:4b:d4:43
```

Vault responds with a generic response as well—key-value pairs. For this path, it responds with the CA's certificate, its expiration date represented in UNIX Epoch time, and the serial number. The certificate shows up twice—once as itself and once as its own `issuing_ca`. The second time has more to do with the structure of Vault's internal code for generating certificates—this will show up later as well. To mirror the last issue, replace the certs/CA.crt file with the certificate PEM block from above.

The last piece of setup in the Vault is that we need to configure roles inside of our `myca` path. These roles are both specific to the `pki` back end and specific to this instance of it. They represent the options for what configurations—namely, the common name, and type—can be put on to the issued certificates. Since we're already using the powerful root token, we're going to generate powerful roles which can mint certificates with any name, but we'll keep separate roles for generating server certificates and client certificates.

```
    shell$ ./vault write myca/roles/powerserver allow_any
_name=true \
        enforce_hostnames=false \
        server_flag=true \
        client_flag=false
    Success! Data written to: myca/roles/powerserver
    shell$ ./vault write myca/roles/powerclient allow_any
_name=true \
        enforce_hostnames=false \
        server_flag=false \
        client_flag=true
    Success! Data written to: myca/roles/powerclient
```

Getting Started: Vault Client Library

To get ready to use the client, we're going to:

- Add a reference client to our project code
- Initialize `dep` and let it pull down our code dependencies

`dep` looks at your code to decide what it needs to pull in. To start to use `dep`, we're going to add the Vault client as an `import` in `certs/generate_certs_vault.go`. Since the package name `api` is a bit too generic, we're going to specify that the qualified identifier of the package name is going to be `vaultapi`.

```
    import (
        vaultapi "github.com/hashicorp/vault/api"
```

Now we let `dep` do the hard part. For demonstration purposes, I'm using the verbose flag; otherwise, `dep` is very quiet.

```
    shell$ dep init -v
    Root project is "github.com/cmceniry/login-glss"
     3 transitively valid internal packages
     2 external packages imported from 2 projects
    (0)   ✓ select (root)
    (1) ? attempt github.com/kelseyhightower/gls with 1 pkgs; 1
versions to try
    (1)     try github.com/kelseyhightower/gls@master
    (1) ✓ select github.com/kelseyhightower/gls@master w/1 pkgs
    (2) ? attempt github.com/hashicorp/vault with 1 pkgs; 76
versions to try
    (2)     try github.com/hashicorp/vault@v0.8.3
    ...
      Locking in master (42a06e0) for direct dep github.com
/kelseyhightower/gls
    ...
      Locking in v0.8.3 (6b29fb2) for direct dep github.com
/hashicorp/vault
      Locking in master (68e816d) for transitive dep github.com
/hashicorp/hcl
```

What is `init` doing?

From our project, it starts by examining every .go file and looking at their `import` statements. From that it starts to build out a list of dependencies and pulls those in. It then does this same examination of the dependencies' `import` statements and iterates. When it looks at a dependency, it looks for any versioning information that that dependency may give—this usually shows up as semantic versioning (`v$major.$minor.$patch`)-based Git tags.

`dep` creates a `Gopkg.toml` file if one does not already exist. The `toml` file is used to specify any version constraints on the dependencies. After `dep` has collected all of the dependency and version information, and any constraints from the `toml` file, it attempts to solve finding the appropriate version of every dependency.

Once solved, `dep` creates a `Gopkg.lock` file, and pulls down any missing dependencies. The `lock` file is the version information which `dep` has picked for the current dependency solution. `dep` uses the `vendor` pattern for storing dependencies. When it pulls down a dependency, it stores that dependency in the vendor directory of this project. This allows the build to be specific to this particular project and not conflate items in the src directory of your `$GOPATH`. That way, if you are working with multiple projects that have conflicting dependencies, you can keep those separate rather than rebuilding your `$GOPATH/src` all of the time.

After `init` is done, it's good to take a look at what it has gathered. The `status` subcommand provides the current dependency solution and also takes a look at the upstream repositories to provide the latest version information.

```
shell$ dep status | cut -b1-80
PROJECT                            CONSTRAINT     VERSION        REVISION LATE
github.com/fatih/structs           *              v1.0           a720dfa  a720
github.com/golang/snappy           *              branch master  553a641  553a
github.com/hashicorp/errwrap       *              branch master  7554cd9  7554
github.com/hashicorp/go-cleanhttp  *              branch master  3573b8b  3573
github.com/hashicorp/go-multierror *              branch master  83588e7  8358
github.com/hashicorp/go-rootcerts  *              branch master  6bb64b3  6bb6
github.com/hashicorp/hcl           *              branch master  68e816d  68e8
github.com/hashicorp/vault         ^0.8.3         v0.8.3         6b29fb2  6b29
github.com/kelseyhightower/gls     branch master  branch master  42a06e0  42a0
github.com/mitchellh/go-homedir    *              branch master  b8bc1bf  b8bc
github.com/mitchellh/mapstructure  *              branch master  d0303fe  d030
github.com/sethgrid/pester         *              branch master  0af5bab  0af5
golang.org/x/net                   *              branch master  0744d00  0744
golang.org/x/text                  *              branch master  1cbadb4  1cba
```

While we won't need it in this exercise, you can always re-solve and bring your dependencies up-to-date with `dep ensure`.

Now that we have our dependencies, we can build out certificate generator.

Generating Keys and Certificates with the Vault API

To start off, our main program needs to set up our Vault connection. The Vault API does this in two parts—initializing the configuration and using that configuration to create a client struct. As with working with the command-line client, we need to explicitly set the Vault address as appropriate.

```
config := vaultapi.DefaultConfig()
config.Address = "http://127.0.0.1:8200"
c, err := vaultapi.NewClient(config)
```

Unlike before, where we set the authentication token as part of our environment, the token is set on the client.

```
c.SetToken("mytoken")
```

Now we can ask Vault to issue a key and certificate. We're going to start with the glssd server certificate.

First, we call `Logical()` to indicate that we're going to be accessing a Vault back end. If we wanted to perform administrative functions, such as `mount`, to Vault instead of data functions, we can use the `Sys()` function to get access there.

As with the command line interface, the `Write` call used a generic interaction with Vault. In the logical subsystem, we call a similar `Write` to the `myca/issue/powerserver` endpoint to have Vault issue us a key and certificate. In addition to the path, we have to supply some data—`common_name` and `ttl` for the expiration to use. How this data is transmitted is where the generic interaction comes into play. The `Write` call of the Vault API has the same signature regardless of what back end is in use. To allow for different back-end forms, it has to rely on loose type checking—the kind you find with the empty interface. And to allow for multiple data parameters, requests to `Write` take data in the form of a map where the map key is the data name as a string, and the map element is the data value as an empty interface.

```
s, err := c.Logical().Write(
  "myca/issue/powerserver",
  map[string]interface{}{
    "common_name": "localhost",
    "ttl":         "1h",
})
```

Vault responds with a Secret struct. There are several parts to it, but the part we care about is in the `Data` field. Much in the same way that the input data was in the generic `map[string]interface{}`, the `Data` field is also a `map[string]interface{}`. We can access the returned keys and assert their type to what we know they are. In particular, for the `pki` back end's issue commands, we get back the `private_key` key and the `certificate`. We take these and assert them to strings. Strings easily cast to byte slices which are what the `ioutil.WriteFile` func needs to save them out to disk.

```
ioutil.WriteFile(
  "certs/server.key",
  []byte(s.Data["private_key"].(string)),
  0444,
)
ioutil.WriteFile(
  "certs/server.crt",
  []byte(s.Data["certificate"].(string)),
  0444,
)
```

Next we do the same actions for the client certificate. This time, we also have to use a different role because that is the role we used which will issue certificates with the client usage set on them.

```
s, err = c.Logical().Write(
  "myca/issue/powerclient",
  map[string]interface{}{
    "common_name": "glss Client A",
    "ttl":         "1h",
  },
)
...
```

```
ioutil.WriteFile(
  "certs/client.key",
  []byte(s.Data["private_key"].(string)),
  0444,
)
ioutil.WriteFile(
  "certs/client.crt",
  []byte(s.Data["certificate"].(string)),
  0444,
)
```

A word of warning: we're being fast and loose with the conversion from the empty interface of the response to something we can write to a file. This means that any issues that crop up here will result in Go panics. It would be advisable to add conversion error checking to this code before building off of it. Or you could use the github.com/mitchellh/mapstructure library, which provides a way to convert loose data into structs much like you would with the encoding/json and encoding/xml libraries.

With that complete, we can run it to generate the new keys and certificates:

```
shell$ go run certs/generate_certs_vault.go
Success!
```

Since we have a drop in replacement for the `generate_certs.go` method, we can run the same commands as we did last issue, and verify that we're still working with the Vault-issued certificates:

```
shell$ ./glssd &
[1] 32659
shell$ 2017/09/23 23:15:31 Starting glsd..
shell$ ./glss .
2017/09/23 23:15:34 user="glss Client A" connect
drwxr-xr-x         442 Sep 23 22:39 .
drwxr-xr-x         510 Sep 23 23:14 .git
-rw-r--r-        42  Sep 23 22:39 .gitignore
-rw-r--r-      2776  Sep 23 22:08 Gopkg.lock
-rw-r--r-       687  Sep 23 22:08 Gopkg.toml
....
```

Application Changes When Using an External Secret Store

To extend the example here, instead of doing a drop-in replacement for the key and certificate generator, you can imagine that the glssd program itself would contact Vault and get a new key and certificate every time it started up. This is interesting for several reasons:

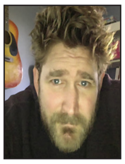
- We know when a specific certificate was issued to a specific client and can track and audit that.
- We can set the certificate lifetimes relatively low, increase key rotation, and decrease the impact timeframe of a leaked key.
- We can apply deployment automation to our environment without having to worry about dirtying our source control systems with keys.

Some of the above is very much dependent on the way the application authenticates to the Vault system, but that will have to wait for a future article. Regardless, the benefits are intriguing.

I hope this article has convinced you that it is relatively straightforward to retrieve items from external secret stores. I hope you take the time to see what they can do for you to help improve the overall security stance in your code and at your organization.

iVoyeur
Tcpdump at Scale

DAVE JOSEPHSEN



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

A little while ago, at my day job, we had a catastrophic DNS outage [1]. DNS (he explained, captain-obviously) is a rather essential bedfellow of SMTP. I'm not understating the relationship because I want to. Honestly, it's sort of impossible to overstate it, or even do justice to how intertwined the protocols really are.

For every SMTP conversation you initiate, you make at least twice the number of DNS queries (and usually, given MX round-robin load balancing, DKIM txt records, and failure-induced retransmission, **way** more). In fact, every time you push that send button to ACK a calendar reply or send a one-liner to thank someone, you're putting more DNS bytes on the wire than you are SMTP.

As an email service provider, sometimes it seems like our **REAL** job is to run DNS at scale. I know, bum-ba-bum *at scale*, those two cheap little words that make every story interesting. You process ACH payment transactions? Okay, that's cool I guess. Oh, you process ACH payment transactions at scale? Why didn't you say so? Any banal undertaking is keynote-worthy if you do it eleventy-billion times a day.

SaaS email service providers, as we are at my day job, carry out SMTP conversations *at scale* at the behest of our customers. All those password reset emails and coupon mailers add up evidently, and the rub, of course, is you can't speak SMTP *at scale* without first speaking DNS *at scale*. It's a very strange business model if you think about it. Step one, implement DynDNS. Step two, layer your actual product on top of it.

As a result, we have a somewhat complicated relationship with AWS in the context of DNS, as you can probably imagine. We often assist them in locating the real-world limitations of this or that service with our perfectly rational, real-world traffic patterns. They often struggle to define the word "abuse" in such a way that doesn't encompass our perfectly rational real-world traffic patterns. We are not unlike a consultant in these respects. An extremely diligent, successful, and annoying consultant.

Were you to ask AWS, I suspect they might tell you that the phrase "perfectly rational" as applied to real-world DNS traffic from a non-DNS provider is subject to interpretation. Of course, they're wrong in our case (with startling regularity), but one does have to respect how predictably on-message they are. For our part, we continue to assure them that gigabyte-sized bursts atop our already absurd cardinality of DNS traffic is a metric of nothing but success for us both, but convincing them of that fact has admittedly been an uphill walk.

We are, however, blessed with a large and seasoned reliability engineering team at SparkPost, so when this latest DNS snafu occurred, there were plenty of eyes and hands at work to mitigate the problem with various temporary nefarious kludges (you know how it is), all of which left me free to poke around what the Internet kids refer to as WTAF.

Now, you're probably thinking to yourself, *ah-hah, this is the point in the story where the auspicious (and ruggedly handsome) author of* ;login: *magazine's column on systems monitoring turns to the highly advanced monitoring platform du jour and, via some arcane means involving machine learning, expertly extracts from it the root cause of the problem*, but alas, no.

I pretty much just launched tcpdump.

Yup, good-ole tcpdump, from the '90s. I wanted to see what was going on on the wire, and, predictably, what was going on was metric tons of DNS transmission failure, but I did learn a few important things:

- The outage was affecting all types of traffic (not just DNS).
- The overwhelming amount of traffic on the wire was outbound DNS traffic to the Internet root NS servers.
- Very little of our outbound DNS traffic was being returned. In fact, we were getting one response for about every 17 queries, but we were getting *a few* responses.

Having been throttled in pretty much every way possible, I thought this looked very much like some sort of throttling, so I wrote this small shell script to broaden my sample set and verify my initial observations.

```
#!/bin/sh
L='<local IP of the machine>'
P=$(tcpdump -c 5000 -vvv -s 0 -l -n)
echo
echo -n "outbound:"
echo "${P}" | grep "${L}.>" | wc -l
echo -n "inbound:"
echo "${P}" | grep "> ${L}" | wc -l
echo -n "top-ten dst"
echo "${P}" | grep '> ..domain' | cut -d\ -f7 | sort | uniq -c |
sort -n | tail -n10 | sort -nr
```

It uses `tcpdump` to sample 5000 packets, measuring the ratio of inbound to outbound packets while dumping a list of the top-10 destination IP addresses. In the grand old days of *actual* computers in *actual* nearby closets, this would have been unnecessary, but I'm sure you've noticed, as I have, just how much of contemporary SRE work consists of convincing upstream service providers not only that a problem exists but that it is, in fact, *their* problem. In this regard, my low-tech shell script was highly successful, and AWS ran off with the baton, only to return some time later to inform us that we'd discovered a new limit in their infrastructure, namely conntrack memory allocations in the VGWs (virtual gateways).

I'll admit, it felt just a tiny bit troglodyte to have, in that time of crisis, turned to tcpdump, but I happen to know that I'm in fine company, because not just one but TWO talks at Monitorama this year had favorable things to say about our loyal old packet-inspecting friend.

Julia Evans' talk [2], entitled "Linux Debugging Tools You'll Love," seemed custom-curated to preach to my personal choir, but really it's Douglas Creager's presentation [3] I'd like to talk to you about.

How shall I phrase this diplomatically? Google builds zany stuff. That's the word. Zany. I feel like that's an arguably uncontroversial, if not objective, observation that we can both agree upon in 2017, and I'm not judging. I sincerely feel like there's nothing *wrong* with their particular brand of zaniness, I mean…it's just what they *do*, and they're very good at it, and it seems to make them happy.

Every time I see a speaker from Google giving a talk on a subject I've never heard of, I scooch down in my seat a little bit to get comfortable, close my laptop lid, and expectantly fold my hands together in preparation for whatever zany systems engineering antics they've gotten up to this time. I know I'm not alone in this. They made a distributed file system over HTTP with 64 MB chunks? Huh. They've overloaded cgroups into a deployment strategy? Cool. They built a compressionless metrics database because they have an infinitely large MapReduce cluster lying around? Okay.

Anyway, my point is, I admit to being a little surprised by Doug's Monitorama talk, wherein he outlines one methodology employed by the "Internetto" team at Google to monitor the edge of Google's network. In that capacity, they've employed visualizations and other analysis tools on good-ole humble libpcap packet captures.

I won't spoil the talk for you, but Doug is a proponent of *continuously* capturing and evaluating TCP headers as a means of monitoring application performance health. In other words, Doug thinks we should all be running tcpdump on every public-facing machine, all the live-long day (though, I suspect he might not phrase it that way). To that end, Google is evidently using service-based libpcap to capture every header of every packet in every end-user interaction on the wire.

Rather than taking a flow-based approach where the sum of all traffic is sampled at a switch, Google runs local pcaps, and RPC ships the data upstream to be centrally processed (no doubt via MapReduce). They extract throughput and latency numbers and do some simple arithmetic, eventually reducing each connection to a JSON blob describing that interaction, complete with Boolean flags like *BufferBloat:true* to indicate common performance problems detected on the wire.

It's an excellent talk which, obviously, should have been called "Tcpdump at Scale," but otherwise was perfect in every way, and if you only have time to see one talk from Monitorama this year, it would be my pick. I'm obviously biased, but I love the approach and sincerely hope it catches on among all sorts of service providers, CDNs, and, especially, IaaS shops that don't have any idea what is transpiring on their networks. Just imagine your cloud provider notifying *you* of wire latency for once (or at least believing you when you report it to them?).

Take it easy.

References

[1] C. McFadden, "How We Tracked Down Unusual DNS Failures in AWS": https://www.sparkpost.com/blog/undocumented-limit-dns-aws/.

[2] J. Evans, "Linux Debugging Tools You'll Love," Monitorama PDX 2017: https://vimeo.com/221062212.

[3] D. Creager, "Packet Captures Are Useful," Monitorama PDX 2017: https://vimeo.com/221056132.

For Good Measure
Letting Go of the Steering Wheel

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

This issue I will talk about Data, not data, that is to say about capital-D Data as the raw material for the world we are now creating. We'll return to working with a small-d data set next time.

I was trained first as an electrical engineer and then as a biostatistician. From engineering, beyond all else the fundamental lesson is that getting the problem statement right is what determines the future, that if you don't get it right then you end up solving a problem you don't have. From biostatistics, beyond all else the fundamental lesson is that all data has bias: the question is whether you can correct for it, and that correcting for data bias in an imperfect world will itself be imperfect. Combining the two, engineering and biostatistics, one is left with two steering questions: where do you actually want to go and what failure modes can you tolerate?

For some time, security training has been both necessary and widely available. The curve of its sophistication and value has been generally upward. We have better tools, we have better understood practices, and we have more and better colleagues. That's the plus side. But I'm interested in the ratio of skill to challenge, and as far as I can estimate, we are expanding the society-wide attack surface faster than we are expanding our collection of tools, practices, and colleagues. If your country is growing more and more food, that's great. If your population is growing faster than your improvements in food production can keep up, that's bad. As with most decision-making under uncertainty, statistics have a role, particularly ratio statistics that magnify trends so that the latency of feedback from policy changes is more quickly clear. Yet statistics require data.

That cybersecurity is hard will come as no surprise, and it has been four years now since the U.S. National Academy of Sciences concluded that cybersecurity should be seen as an *occupation* and not a *profession* because the rate of change is too great to enable professionalization [8]. That rate of change is why cybersecurity is perhaps the most intellectually demanding occupation on the planet, and it may well be that the hybrid vigor of retreading other professions, other skill sets for cybersecurity practice has been and remains crucial to cybersecurity outcomes rather than a random bit of historical trivia.

Winston Churchill said, "The further back I look, the further forward I can see." Churchill was arguing for looking back centuries so as to discern the patterns of human affairs, to find commonalities within the dynamics of competition at whatever scale fit the age in which those competitions occurred so as to see forward and win the then current competition. But should we measure time in constant units—a day, a week, a month, a year—or should it be something akin to a log scale denoted not by the rate at which the seconds pass on a constant clock but by the number of events that have passed? Does a rapid rate of change mean we only have to look back a littler bit in chronologic time but further back in ever-denser event logs? Or must we look back further still both in time and event counts if we are to damp out the noise of the present?

When I look back to earlier stages of my own career, the principal difficulty of any particular stage has oscillated between getting the problem statement right and picking the failure mode that is tolerable given what data was available on which to make a decision. I don't think that has changed. As of today, data acquisition wouldn't seem to be the problem insofar as instrumentation is cheap and mostly reliable. But data has to be collected with an hypothesis in mind, or, as Charles Darwin said, "All observation must be for or against some view if it is to be of any service." That brings us back to the problem statement, that is to say what problem are you trying to solve and, therefore, what data do you need to collect to have it be of any service?

To repeat, you need to know something about what problem you are trying to solve and what data would help you make the decisions that solve that problem. Over a small number of years, the term "data science" has become commonplace. It seems first to have been used over 50 years ago, but the current usage stems most directly from a 1997 lecture by Jeff Wu with the title "[Does] Statistics=Data Science?" [5]. Wu characterized statistical work as a trilogy of data collection, data modeling, and decision-making. In his lecture's conclusion, he initiated the modern usage of the term "data science" and advocated that statistics be renamed data science and statisticians be renamed data scientists. Those semantics seem to add little clarity to what the collection, modeling, and use of data provide, but argument over terminology is a hallmark of how a science develops.

But Darwin's remark that all observation must be for or against some view is not quite right, at least not quite right for us here. It is not so simple as Wu's data collection, data modeling, and decision-making, either. When you collect data and with it build a model, your goal, your problem statement, matters. If your purpose in building a model is to come to a definitive conclusion about causality, about how nature works, then you are saying that the inputs to your model and the coefficients that calibrate their influence within your model are what matters in the final analysis. Parsimony in the sense of Occam's Razor is your judge, or, as Antoine de Saint-Exupéry put it, "You know you have achieved perfection in design, not when you have nothing more to add, but when you have nothing more to take away." So it is when you are chasing causality.

By contrast, when your purpose in building a model is to enable control of some process or other, then you will not mind if your input variables are correlated or redundant—their correlation and their redundancy are not an issue if your goal is to direct action rather than to explain causality.

In some circumstances you can do both, that is you can both explain causality and enable control. In those situations, it is your model's ability to predict that both satisfies the reader that you have captured a causal relationship and that operationalizes the model's predictions irrespective of any underlying truths [7]. A goal of understanding causality in its full elegance leads to F=ma or E=mc^2. A goal of control leads to econometric models with thousands of input variables each of whose individual contribution is neither clear nor relevant.

Consider anomaly detection and its role in current cybersecurity products. Anomaly detection presumes something about distributions of detectible events, namely that within a selected interval anything outside some bounding box is worth investigation. It is not concerned with causality; it is concerned with control irrespective of causality. This is a coherent strategy, though with side effects.

Or consider "Big Data" and deep learning. Even if Moore's Law remains forever valid, there will never be enough computing, and hence data-driven algorithms must favor efficiency above all else as data volume grows. Yet the more efficient the algorithm, the less interrogatable it is, that is to say that the more optimized the algorithm is, the harder it is to know what the algorithm is really doing. That was the exact theme of a workshop held in New York by Morgan Stanley and the Santa Fe Institute three Octobers ago titled, "Are Optimality and Efficiency the Enemies of Robustness and Resilience?"

The more desirable some particular automation is judged to be, the more data it is given. The more data it is given, the more its data utilization efficiency matters. The more its data utilization efficiency matters, the more its algorithms will evolve to opaque operation. Above some threshold of dependence on such an algorithm in practice, there can be no going back. As such, preserving algorithm interrogatability despite efficiency-seeking, self-driven evolution is the pinnacle research-grade problem that is now on the table, and I mean for all of cybersecurity. If science does not pick up this challenge, then Larry Lessig's characterization of code as law is fulfilled. A couple of other law professors have seized on that very idea and suggested that price-fixing collusion among robots will be harder to detect than collusion among people [12].

The point is this: if we choose control as the purpose of our efforts, then we will have to let causality become harder to see because our models will submerge any causal relationships in a thicket of confounding. If, instead, we focus on causality, the very things that we need to measure become harder to get if, for no other reason, our sentient opponents will make it so. I'm for measurement as decision support, i.e., I am in the control camp, not the causality camp. At the same time, I very much do demand that I be able to ask some algorithm, "Why did you do that?" and get a meaningful answer. Overall, having both control and interrogatability is a difficult problem to say the least.

And that may be the most important thing I have to say here, that the real problem statement is not about cybersecurity per se but about the side effects of our pursuit of it. Some years ago, in a lecture at Harvard's Kennedy School of Government, the speaker listed the Four Verities of Government as:

- Most exciting issues are not important.
- Most important issues are not exciting.
- Not every problem has a good solution.
- Every solution has side effects.

I think that those are the verities of cybersecurity, too. The din of press coverage of cybersecurity is only about the exciting failures, not the important successes nor that even more important trendline for the ratio of skill to challenge. Perhaps this simply reinforces Donald Knuth's remark that "Premature optimization is the root of all evil." Perhaps it is simply that evolution in our digital world follows the same patterns as evolution in the natural world.

If that is so, then what we see in Nature is what we should expect to see in cybersecurity. Well, in Nature there are two alternative games for survival, r-selection and K-selection [2]. R-selected species produce many offspring, each of whom has a relatively low probability of surviving to adulthood, while K-selected species are strong competitors in crowded niches. K-selected species invest more heavily in much fewer offspring, each of whom has a relatively high probability of surviving to adulthood. If we change the term from "produce many offspring" to "re-image frequently" you now have precisely the world of VMs. Or, to be more current still, you have the kind of components in a DevOps setting where it is arguable whether moving target defense or minimizing new product introduction latency is the paramount goal or value.

Stephen Jay Gould's idea of punctuated equilibrium [4] as the fundamental cadence of evolution has a hold on me. In his formulation, long periods of stasis are the norm. In computing, we would call that "legacy." I trace the birth of the cybersecurity industry to Microsoft's introduction of a TCP/IP stack as a freebie in the Windows 95 platform, thereby taking an operating system designed for a single owner/operator on a private net, if any, and connecting it to a world where every sociopath is your next door neighbor. That event was the birth of our industry, though the fact was unnoticed at the time.

The second of these punctuations occurred around a decade ago when our principal opponents changed over from adventurers and braggarts to professionals. From there on, mercenaries, some armed with zero-days, dominated. The defense response has been varied, but the rise of bug-bounty programs and software analysis companies are the most obvious. An Internet of Things (IoT) with a compound annual growth rate of 35% will be like anabolic steroids for at least those two.

In August 2016, we passed a third such punctuation. The DARPA Cyber Grand Challenge [1] showed that what has heretofore required human experts will shortly come within the ken of fully automatic programs, or, shall we say, algorithms that are today at the upper level of skill, with intelligence, per se, soon to follow. As with both of the previous two punctuations, the effects of the third will reverberate for the indefinite future. I have long argued that all security technologies are dual use, and the day after the Cyber Grand Challenge, Mike Walker, its DARPA program manager, said as much: "I cannot change the reality that all security tools are dual-use."

And everywhere the talk is about "Big Data" and how much better an instrumented society will be. The cumulative sum of the curves for computing, storage, and bandwidth is this: in 1986 you could fill the world's total storage using the world's total bandwidth in two days. Today, it would probably take nine months of the world's total bandwidth to fill the world's total storage [6], but because of replication, synchronization, and sensor-driven autonomy, it is no longer really possible to know how much data there is. Decision-making that depends or depended on knowing how much data there is is over.

In other words, whatever the future holds, it is clear that it will be data rich and that the tools acting on it will be dual use. The classic triad of cybersecurity has long been confidentiality, integrity, and availability, and we have heretofore prioritized confidentiality, especially in the military sector. That will not be the case going forward, and not just because the rising generations have a relaxed complacency about the tradeoffs in information sharing between what it enables and what it disables. In the civilian sector, integrity will supplant confidentiality as the pinnacle goal of cybersecurity. In the military sector, weapons against data integrity already far surpass weapons against data confidentiality.

This trend—the eclipse of confidentiality by integrity and availability—is solidly entrenched now. Already algorithms learn rather than being taught. What they learn depends on how their learning is scored. This is behavioral reinforcement of a form that would be entirely familiar to B. F. Skinner—you don't teach the subject the desired behavior, you reward the subject for exhibiting the desired behavior. You don't look into the mind of the human subject nor into the structure of the self-modifying algorithm, you just look at the objective reality of behavior itself. This is not so much our creation of an intelligence but an unforced assumption that an intelligence will appear if given enough training sets.

But because of how that kind of learning works, it can be fooled by data as easily as it can be enriched by it. In a 2013 paper, Szegedy et al. found that

> [D]eep neural networks learn input-output mappings that are fairly discontinuous to a significant extent. We can cause the network to misclassify an image by applying a certain imperceptible perturbation, which is found by maximizing the network's prediction error. In addition, the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the dataset, to misclassify the same input [10].

Not even a year ago, researchers in France and Switzerland found that

> Given a state-of-the-art deep neural network classifier, we show the existence of a *universal* (image-agnostic) and very small perturbation vector that causes natural images to be misclassified with high probability. We propose a systematic algorithm for computing universal perturbations, and show that state-of-the-art deep neural networks are highly vulnerable to such perturbations, albeit being quasi-imperceptible to the human eye. We further empirically analyze these universal perturbations and show, in particular, that they generalize very well across neural networks. The surprising existence of universal perturbations reveals important geometric correlations among the high-dimensional decision boundary of classifiers. It further outlines potential security breaches with the existence of single directions in the input space that adversaries can possibly exploit to break a classifier on most natural images [11].

Note to reader: look up "adversarial perturbations."

So why am I making a point about image classification by deep neural networks? Because it raises the fundamental question: given data richness and self-modifying algorithms becoming ever more prevalent in the cybersecurity regime, is keeping a human in the loop a liability or a failsafe? I've already written that the central requirement for security is keeping a human in the loop, that of interrogatability. But there are others, not the least of which is reaction time.

Nevertheless, as data volume grows it creates a challenge far beyond the parlor exercise of how long would it take to fill all the world's storage with all the world's bandwidth, yet it is bandwidth itself that is a limiting coefficient. It is safe to predict that the F-35 will be the last manned fighter plane; drone fleets make more sense going forward. Those drone fleets require ever more massive compute power handling, ever more massive data flows. Lt. Colonel Rhett Hierlmeier heads up the training center for the F-35. He believes that what is today a training simulator will tomorrow be a control point, not a simulator. *Popular Science*'s interview with him includes this telling snippet: "Standing outside the cockpit, he peers into the darkened dome, and says he believes we will one day fight our enemies from inside one of these things. When I ask what that will take, he says flatly, 'Bandwidth'" [9]. Just that point about bandwidth is why "engineers are focused on things like improving artificial intelligence so planes can act with more autonomy, thus cutting down on communication bandwidth."

And the same thing will apply in our field. My estimate is that the Internet of Things has a 35% compound annual growth rate. If I am approximately correct, then IoT growth is already outdistancing the growth rate for installed bandwidth, and for us as much as for fighter pilots the pressure for autonomy is and will be driven by the data-sensing capacity of a rapidly increasing installed base.

Let me therefore suggest that when sentience is available, automation will increase risk, whereas when sentience is not available, automation can reduce risk. Note that parsing, that replacing available sentience with something that is not sentient *will* increase risk but that substituting automation for whatever you have absent sentience *can* make things better. It won't do so necessarily, but it can.

As a child of the hillbilly South, I have nothing against automating away drudgery; a 110-year-old woman interviewed for the book *Supercentenarians* was asked what was the most important invention during her lifetime. Her answer was the washing machine. But with the spread of computers, we have tended to use automation as soon as it is cheaper than human labor. No single replacement of labor by automation matters, but the sum of it does. Yet as we sit here today, the equation of automation is not that of eliminating drudgery but eliminating the need for sentience. Is there enough available sentience to indict cybersecurity automation as risk creating or, alternately, is there far too little sentience that is up to the task at hand and therefore automation is essential and risk reducing?

The embedded systems space has long since made the attack surface of the non-embedded space trivial by comparison. It was two years ago when the count of networked devices exceeded the count of human beings [3]. Qualcomm's Swarm Lab at UC Berkeley predicts 1000 radios per human by 2025, while Pete Diamandis' *Abundance* calls for 45x10^{12} networked sensors by 2035. These kinds of scale cannot be supervised, they can only be deployed and left to free-run. If any of this free-running is self-modifying, the concept of attack surface is just plain over as is the concept of trustworthy computing, at least as those are presently understood. This will echo John McAfee's April 2017 interview in *Newsweek*: "Any logical structure that humans can conceive will be susceptible to hacking, and the more complex the structure, the more certain that it can be hacked."

So the situation with data in cybersecurity is richly complex. We need ever more of it if we are to capture increasingly subtle attack vectors, and especially so if we want autonomous, learning-capable algorithms that need to be faster than we are or which don't have the bandwidth to tell us what they are seeing. Yet the more important the decision to be made, the more vital it is to keep a human in the loop.

That is a tall problem statement, and to go with it we need to carefully consider what the tolerable failure modes are. Do we want to trust no sensor data that can't be corroborated? Do we want to accept algorithms as better managers than we are even when we can't tell how it is that they do what they do? Do we want to keep humans in the loop and, if so, how do we protect their legal culpability when it can be shown that some algorithm would not have made mistakes as costly as the ones the human made?

I urge you to take in data that you have some feel for, that is to say for which you have at least some calibrated understanding such that your presence in the loop is prima facie meaningful. If you are designing algorithms, work hard on making them interrogatable. If you are of necessity relying on self-modifying algorithms, let Santayana remind you that "Skepticism is the chastity of the intellect." Remember that all data has bias and that that, too, is in the equation for what failure modes you can tolerate.

References

[1] DARPA Cyber Grand Challenge, August 4, 2016: http://archive.darpa.mil/cybergrandchallenge/.

[2] E. Pianka, "On r and K Selection," *American Naturalist,* vol. 102 (1970), pp. 592–597: http://bit.ly/2fmrZf8.

[3] D. Geer, "For Good Measure: Implications of the IoT," *;login:,* vol. 41, no. 4 (December 2016): geer.tinho.net/fgm/fgm.geer.1612.pdf.

[4] N. Eldredge and S. J. Gould, "Punctuated Equilibria: An Alternative to Phyletic Gradualism," in *Models in Paleobiology* (Freeman Cooper, 1972), pp. 82-115: www.blackwellpublishing.com/ridley/classictexts/eldredge.asp.

[5] www2.isye.gatech.edu/~jeffwu/presentations/datascience.pdf as drawn from Wikipedia, "Data Science": en.wikipedia.org/wiki/Data_science.

[6] M. Hilbert, "World's Information Capacity PPTS": www.martinhilbert.net/WorldInfoCapacityPPT.html (reflecting M. Hilbert & P. Lopez, *Science,* vol. 332, no. 6025 (2011), pp. 60–65) extrapolated with concurrence of its author; see also http://bit.ly/2hmHArN.

[7] "You see, there is only one constant, one universal. It is the only real truth. Causality."—Merovingian in *The Matrix Reloaded.*

[8] National Research Council, "Professionalizing the Nation's Cybersecurity Workforce? Criteria for Decision-Making": www.nap.edu/openbook.php?record_id=18446.

[9] K. Gray, "The Last Fighter Pilot," *Popular Science,* December 22, 2015: www.popsci.com/last-fighter-pilot.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, "Intriguing Properties of Neural Networks," February 2014: arxiv.org/pdf/1312.6199.pdf.

[11] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, P. Frossard, "Universal Adversarial Perturbations," October 2016: arxiv.org/pdf/1610.08401v1.pdf.

[12] S. Farro, "When Robots Collude: Computers Are Adopting a Legally Questionable Means to Crush the Competition" (algorithms can learn to do so), *Business Insider,* April 28, 2015: http://read.bi/2feNmuW.

/dev/random
Cloudbursting, or Risk Mismanagement

ROBERT G. FERRELL

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

As I write this, my area is still recovering from Hurricane Harvey, while Floridians are sloshing their way out from under Irma. People who choose to live in places like this are perfectly well aware that sooner or later we're going to get smacked by a tropical weather system, however. It's a risk we all accept and manage—with varying degrees of skill.

I built my former career on risk and the mitigation thereof. Well, at least the second half of it or so. Before that I was an analytical chemist, colon cancer researcher (which largely entailed cleaning up radioactive rat poop), percussionist, professional grad school dropout (three different programs), and probably some other stuff I don't remember. But (as I expect it will say on my epitaph), I digress. (Sorry for all the asides: I had some parentheses in stock that had reached their "use-by" date. Waste not, want not.)

We all take risks. Heck, just by reading my column you're risking long-term neocortical damage. If you don't work from home (or, if your house is as cluttered as mine, even then), you're taking a substantial risk just traveling to your place of employment. Yes, really. Have you seen the way some of those people out there drive? The other day I spotted a doofus sitting in the driver's seat of a big truck while yakking on the cell phone and apparently watching a DVD on the little screen perched on his dashboard. (I hesitate to say he was "driving" the truck, because I don't think that's an accurate assessment.) Maybe it was a defensive driving video, I don't know. Regardless, it did not appear that keeping his vehicle between the lines, or even on the asphalt, was high on his priority list. I just pulled over to a rest area for a few minutes to give him time to have his accident without me. Not that you could really call it an "accident": more like a "grossly negligent."

Horrifyingly bad drivers illustrate the category of existential risk we can't realistically avoid. Another member of this set is what I refer to as a "cloudburst," or loss of personal data entrusted to some third party without the owner's explicit awareness. No matter how diligently you ensure that your connection to a first-line merchant or financial provider is protected by SSL/TLS/whatever with valid certificates, you have zero control or even in most cases cognizance of what happens to that precious information once you've transferred it thusly. It is now completely at the mercy of any thief who manages to defeat the safeguards of nebulous third parties. It's somewhat like carrying your cash to the bank in an armored car only to have them store it in the lobby, "protected" by cardboard boxes marked Please Do Not Steal.

If you decide to go that extra diligence mile and track your data beyond the first step, good luck. Most institutions are extremely reticent when it comes to sharing details of their processing chain, for "security" reasons. That is, they don't want you to know that there really isn't much of that going on. "That's not something you should worry your pretty little head about," they'll say condescendingly, "We've got it under control," and give you a big thumbs up. The next week you receive an email informing you that your account was one of 200 million compromised—and here's a year of free credit monitoring, not that it will do you much good. I think I've had free credit monitoring for almost a decade now because of these little serial overlapping "security incidents," and all I've gotten out of it were a long string of warnings and vague reassurances.

The "cancel every credit instrument and change all 85 of your online passwords" circus is getting to be far too routine for my taste. Some of my credit cards are on their fourth or fifth iteration, all because various financial processing entities along the path of ignominy couldn't keep their security diapers pinned.

What is the fundamental malfunction with these firms/agencies? Why are they taking so many liberties with our precious data? I don't know: hubris, maybe, or perhaps our old scabrous nemeses ignorance and indolence. Whatever the case, compromise is the new norm. A foreign government lacking even the pretense of having my best interests at heart, for example, now possesses the volumes of incredibly intrusive information I supplied to get the security clearance I held in my former career. The irony of a government that can't keep its own barn door shut questioning me at length about my ability to preserve secrets would be laughable if it weren't so egregious. At least when I got read onto a SAP (Special Access Program), I didn't turn around and store the relevant info on Dropbox with the password "Unc73$4M," 'leet-speak for Uncle Sam.

So, what can we, as information technology professionals, do about this—apart from posing largely rhetorical questions? While constantly increasing both encryption key lengths and the complexity of passwords may give senior management, stockholders, and legislators a warm fuzzy, the real answer lies in educating the people along that data processing path. Technology, regardless of how well designed or robustly implemented, cannot take the place of human security awareness. These protocols and hardware devices and algorithms are only as effective as the people who deploy them. It is evident that if we don't change the way rank and file employees think about and implement data security, any reasonable expectation of identity fidelity is but a fool's dream. No matter how sharp your plow or powerful your oxen, the furrows you dig will not yield any crops if the soil itself is poor.

(Mmm. Pre-industrial agriculture analogies always make me hungry. Fortunately, I keep a pot of gruel going next to my computer for such contingencies.)

The state of Illinois is reportedly working on employing blockchains to provide a "sovereign digital identity." I applaud this effort and any others that help us move away from static identifiers like Social Security numbers that, once compromised, become liabilities rather than authenticators. They are, in effect, lifelong passwords you can't (easily) change.

If you are one of those third-party data-manglers, my suggestion for storage of personally identifying information is to use a randomized multicontainer approach. With SSNs, for example, you could split the numbers into chunks of two digits and then randomly store each chunk in one of five containers. You could use the same algorithm with name or other data fields, actually, except that the string length would be variable. Unique identifiers would therefore be serial numbers containing the location and position within the string (and in some cases, length) of each chunk. You'd have to map the positional data for name and SSN in a relational database of some kind, of course, but your network folks already more or less do that with NAT, so that will be familiar ground. You could even craft said UIDs in the same format as SSNs, to confuse thieves into thinking they've stolen the data, rather than the pointer.

Or you could just stop opening attachments and using Pirate Bay torrents at work. I don't want to suggest anything too outlandish and ridiculous, however. Even humorists have their limits.

Book Reviews

MARK LAMOURINE AND MICHELE NELSON

Gnuplot in Action: Understanding Data with Graphs (Second Edition)
Phillipp K. Janert
Manning Publications Co., 2016, 372 pages
ISBN 978-1-63343-018-1
Reviewed by Mark Lamourine

Gnuplot has always been a chicken/egg tool for me. There are times when I have data series I'd like to plot, but I'm not fluent enough with Gnuplot to produce something useful quickly. Without interesting (read: urgent) data, I have more pressing things to do than to learn a new tool. When I saw that there was a new edition of *Gnuplot in Action* I thought it was time to try again.

Gnuplot is unrelated to the GNU project, but it adheres strongly to the UNIX ideal to do one thing well. It takes primarily flat text files and produces only 2D x-y plots of the data. It is designed to allow interactive operation and very simple batch scripted "programs."

Janert writes with the same philosophy, taking advantage of the interactive nature of Gnuplot to get the reader started producing plots immediately. By the end of the fourth chapter, I had everything I needed to create clean, simple two-variable plots that would easily serve the uses I have had in the past. The only thing I wish were offered earlier in the book is date and timestamp parsing. That had to wait for Chapter 8.

Gnuplot does have a number of shortcomings that result from this philosophical simplicity. All variables are global. The control structures and function definition syntax are rudimentary. The functionality that a user of modern scripting environments might expect (locally scoped variables?) just doesn't exist or is simulated with what can charitably be called hacks. Janert doesn't shy away from these limitations but, rather, explains the reasoning that has led to them and then shows how to make the most of what is there.

In the latter half of the book, Janert deals with scripting, streaming data, and even animation. These aren't things I ever expect to do with Gnuplot, but it's interesting to know that you can. In all of these areas his writing is to the point and clear. The next time I have a data series I need to visualize, I know now that I'll be able to get some quick clean plots.

Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People
Aditya Y. Bhargava
Manning Publications Co., 2016, 238 pages
ISBN 978-1-61729-223-1
Reviewed by Mark Lamourine

Grokking Algorithms is a breezy, comfortable introduction to computer algorithms. I was originally attracted to it because of the *grok* in the title, a term coined by Robert Heinlein in *Stranger in a Strange Land* and adopted by the computer community by the 1980s to mean *to understand deeply*. While I've heard and used the term, I've never before seen it in the title of a book and wondered whether any book could aspire to help someone grok anything. I was also intrigued by the subtitle, *An Illustrated Guide for Programmers and Other Curious People*. Are there non-programmers who are curious about algorithms, and how would one approach algorithms with them?

Bhargava is both the writer and illustrator. He writes and uses his drawings to clarify the text. His drawings feature a pen-and-ink style that is much softer than the typical spare computer-generated graphics so common today. This, along with his inclusive narrative style, gives his book a living-room feel that is in sharp contrast to most technical writing being done.

This isn't an academic tome—though it might be used as a high-school or freshman college intro to algorithms course—and Bhargava isn't a theorist; he's an artist by training and a practical coder by profession. His goal is to give the reader a sense of how algorithms are created and how they work. He offers one of the better explanations of Big O notation that I've seen. He also makes a point that I think is often missed, that two algorithms of the same order can still have very different efficiency if the cost of a single cycle is higher than the other.

There are algorithms here that didn't exist or were not in common use when I studied. Bhargava devotes a chapter each to "Dynamic Programming" and "Greedy Algorithms," and applies the K-Nearest Neighbors algorithm to a simple OCR problem. In the closing chapter, he just mentions 10 more algorithms, talking about what they are used for and why they are important. These include a one-page exposition of tree algorithms, a few pages on MapReduce, and a page each on SHA hashing, Diffie-Hellman key exchange, and linear programming.

Even where Bhargava goes into more depth about how specific algorithms work, he's not all that concerned with implementation details. It's especially true when he talks about hash tables and sorts. He does go into enough detail to help the reader understand when and how to use a hash table, but he explains, correctly, that most modern languages have some hash table feature, and that most coders will never have to implement a hash table.

Reading *Grokking Algorithms* won't make you a programmer, but I really like it as an introduction for someone who is curious about why and how people solve problems with software. I do think the motivated reader will go away with a deeper understanding of how computers are used to ask and answer questions.

For Fun and Profit: A History of the Free and Open Source Software Revolution
Christopher Tozzi
The MIT Press, 2017, 324 pages
ISBN 978-0-262-03647-4

Reviewed by Michele Nelson

I volunteered to read and review Christopher Tozzi's *For Fun and Profit* because of my interest in the history of computing and how the tools I use today came to be. It turns out that I have lived through a revolution without even realizing it.

In his lengthy introduction, Tozzi, an Assistant Professor of History at Howard University, offers the theory that what he refers to as the free and open source software (FOSS) revolution that began in the 1980s has much in common with the French Revolution of 1789. The French Revolution sought to achieve liberty, fraternity, and equality. The FOSS revolution sought to restore software freedom. Both changed the world.

I always thought that "free software" referred to any software you don't have to pay to use, and that "open source" meant software you don't have to pay to use that also has the source code available at no cost. According to Tozzi, however, FOSS is not so easy to define. He feels that both "free software" and "open source" are ambiguous terms. After much discussion in the introduction, he defines "free software," for the purposes of this book, as software "whose programmers or users call it such because its source code can be studied and modified freely by people who use the software, whether or not the source code costs money." He uses the term "open source" to refer to software "whose creators and users preferred that term over 'free software.'" That still seems a bit ambiguous to me.

The book is divided into six chapters, tracing the FOSS revolution from its beginnings to its current status. Tozzi begins with a discussion of the origins of hacker culture, defining the word "hacker" as "the class of programmers who espouse the hacker ethic" and devoting several pages to the definition of the "hacker ethic." He goes on to tell the story of the birth of the BSD and GNU operating systems, devotes a chapter to "The Story of Linux," and also covers Richard Stallman's GNU Project and the Free Software Foundation in detail. His story continues with the "moderate phase" of the revolution—Linux and GNU distributions, office apps, email, and Web—Apache, Samba, MySQL, and PHP.

The chapter titled "The FOSS Revolutionary Wars" was the most interesting to me. According to the author, there were two wars being fought: one inside the FOSS community—"The FOSS Civil War"—and another that pitted the FOSS community against proprietary software companies, chiefly Microsoft.

Tozzi describes the battle within the FOSS community as a fight over what "free software" and "open source" actually meant. He concludes that this battle was not won by either side and is still a contentious issue for some. At the same time as this conflict was causing a rift in the community, there were external threats that required them to band together despite their disagreements. As Tozzi explains it, proprietary software companies, especially Microsoft, were getting increasingly nervous as FOSS products began to be accepted, even endorsed, in the business world. When Netscape released the Mozilla browser as an open source product in 1998, Microsoft went on the offensive. The story of that battle, involving leaked internal Microsoft documents, lawsuits and countersuits, and a report from an outfit called the Alexis de Tocqueville Institution contending that Linus Torvalds "must have plagiarized much of the Linux source code" from Andrew Tanenbaum's Minix operating system, is one of the best parts of this book.

In the final chapter, Tozzi describes how FOSS has evolved since the early 2000s, looking at some key developments: the endorsement of FOSS by many large companies, including Microsoft; the Android mobile OS; Ubuntu/Linux; and cloud and embedded computing. This chapter also includes a discussion of the "free culture" movement, citing Creative Commons and Wikipedia as two examples of projects that have "extended FOSS principles into new territory." He ends with a discussion of diversity, or rather the lack of it, in the software industry as a whole and in the FOSS community in particular.

In *For Fun and Profit,* Christopher Tozzi relates the history of the FOSS revolution from the origins of hacker culture through the end of the revolutionary wars to the present. Some of the chapter introductions are rather tedious, and the writing could be livelier, but overall, I found this to be an interesting read that I will no doubt revisit.

Statement of Ownership, Management, and Circulation, 10/01/2017

Title: *;login:* Pub. No. 0008-334. Frequency: Quarterly. Number of issues published annually: 4. 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 (Fall 2017) Published Nearest to Filing Date
a. Total Number of Copies			2587	2400
b. Paid Circulation	(1)	Outside-County Mail Subscriptions	1192	1108
	(2)	In-County Subscriptions	0	0
	(3)	Other Non-USPS Paid Distribution	738	725
	(4)	Other Classes	0	0
c. Total Paid Distribution			1930	1833
d. Free Distribution By Mail	(1)	Outside-County	0	0
	(2)	In-County	76	77
	(3)	Other Classes Mailed Through the USPS	35	37
	(4)	Free Distribution Outside the Mail	306	125
e. Total Free Distribution			417	239
f. Total Distribution			2347	2072
g. Copies Not Distributed			240	328
h. Total			2587	2400
i. Percent Paid			82%	88%
Paid Electronic Copies			512	490
Total Paid Print Copies			2442	2323
Total Print Distribution			2859	2562
Percent Paid (Both Print and Electronic Copies)			85%	91%

I certify that the statements made by me above are correct and complete.

Michele Nelson, Managing Editor 10/1/17

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

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

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

Discounts on registration fees for all USENIX conferences

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

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

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

USENIX Board of Directors

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

2018 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 2018. A report from the Nominating Committee is now available on the USENIX Web site at www.usenix.org/board/elections18. USENIX members will also receive notification of this report via email.

Nominations from the membership are open until January 2, 2018. To nominate an individual, send a written statement of nomination signed by at least five 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 2, 2018. Please prepare a plain-text Candidate's Statement and send both the statement and a 600 dpi photograph to production@usenix.org, to be included on the ballots.

In early February 2018, ballots will be mailed to all members in good standing. Ballots must be received in the USENIX offices by March 30, 2018. The results of the election will be announced on the USENIX Web site by April 11 and will be published in the Summer 2018 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, 2018, whichever is earlier.

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 2017.

Program Chairs

Enigma 2017
David Brumley and Parisa Tabriz

15th USENIX Conference on File and Storage Technologies (FAST '17)
Geoff Kuenning and Carl Waldspurger

2017 USENIX Research in Linux File and Storage Technologies Summit (Linux FAST Summit '17)
Christoph Hellwig and Ric Wheeler

SREcon17 Americas
Kurt Andersen and Liz Fong-Jones

14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)
Aditya Akella and Jon Howell

SREcon17 Asia/Australia
Tammy Butow and Jun Liu

2017 USENIX Annual Technical Conference (USENIX ATC '17)
Dilma Da Silva and Bryan Ford

9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '17)
Marcos Aguilera and Angela Demke Brown

9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '17)
Eyal de Lara and Swaminathan Sundararaman

Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)
Sonia Chiasson and Matthew Smith;
Mary Ellen Zurko (General Chair)

26th USENIX Security Symposium (USENIX Security '17)
Engin Kirda and Thomas Ristenpart

2017 USENIX Summit on Hot Topics in Security (HotSec '17)
David Brumley and Parisa Tabriz

2017 USENIX Workshop on Advances in Security Education (ASE '17)
Mark Gondree and Ashley Podhradsky

10th USENIX Workshop on Cyber Security Experimentation and Test (CSET '17)
José M. Fernandez and Mathias Payer

11th USENIX Workshop on Offensive Technologies (WOOT '17)
William Enck and Collin Mulliner

7th USENIX Workshop on Free and Open Communications on the Internet (FOCI '17)
Jon Penney and Nicholas Weaver

SREcon17 Europe/Middle East/Africa
Avishai Ish-Shalom and Laura Nolan

31st Large Installation System Administration Conference (LISA17)
Caskey Dickson and Connie-Lynne Villani

Other Chairs and Major Contributors

FAST '17
Work-in-Progress/Posters Co-Chairs: Irfan Ahmad and Theodore Wong

Tutorial Coordinators: John Strunk and Eno Thereska

NSDI '17
Poster Session Co-Chairs: Anirudh Badam and Mosharaf Chowdhury

SOUPS 2017
Invited Talks Chair: Robert Biddle

Lightning Talks and Demos Chair: Heather Crawford and Elizabeth Stobert

Karat Award Chair: Dave Crocker

Posters Co-Chairs: Michelle Mazurek and Kent Seamons

Tutorials and Workshops Co-Chairs: Adam Aviv and Florian Schaub

Publicity Chair: Patrick Gage Kelley

USENIX Security '17
Invited Talks Committee: Michael Bailey, David Molnar, and Franziska Roesner

Poster Session Chair: Nick Nikiforakis

Test of Time Awards Committee: Matt Blaze, Dan Boneh, Kevin Fu, and David Wagner

Lightning Talks Co-Chairs: Kevin Butler and Deian Stefan

Enigma Interviews
Seth Rosenblatt and Sean Sposito

LISA17
Invited Talks Co-Chairs: Pat Cable and Alice Goldfuss

Tutorial Co-Chairs: Mike Ciavarella and Courtney Eckhardt

LISA Lab Coordinators: Branson Matheson and Brett Thorson

Storage Pavilion and Data Storage Day at LISA17
Organizer: Jacob Farmer of Cambridge Computer

USENIX Board of Directors
Cat Allman, Michael Bailey, David Blank-Edelman, Angela Demke Brown, Daniel V. Klein, Kurt Opsahl, Carolyn Rowland, and Hakim Weatherspoon

Audit Committee
Eric Allman, John Arrasjid, and Niels Provos

HotCRP Submissions and Reviewing System
Eddie Kohler

USENIX ASSOCIATION FINANCIAL STATEMENTS FOR 2016

The following information is provided as the annual report of the USENIX Association's finances. The accompanying statements have been prepared by Bong Hillberg Lewis Fischesser LLP, CPAs, in accordance with Statements on Standards for Accounting and Review Services issued by the American Institute of Certified Public Accountants. The 2016 financial statements were also audited by Bong Hillberg Lewis Fischesser LLP. Accompanying the statements are charts that illustrate the breakdown of the following: operating expenses, program expenses, and general and administrative expenses. The Association's operating expenses consist of its program; management and general; and fundraising expenses, as illustrated in Chart 1. These operating expenses include the general and administrative expenses allocated across all of the Association's activities. Chart 2 shows USENIX's program expenses, a subset of its operating expenses. The individual portions shown represent expenses for conferences and workshops; membership (including *;login:* magazine); and project, program, and good works. Chart 3 shows the details of what comprises USENIX's general, administrative, and management expenses. The Association's complete financial statements for the fiscal year ended December 31, 2016, are available on request.

Casey Henderson, Executive Director

USENIX ASSOCIATION

Statements of Financial Position
December 31, 2016 and 2015

	2016	2015
ASSETS		
Current assets		
Cash and equivalents	$ 742,910	$ 431,293
Accounts receivable	94,535	355,919
Prepaid expenses	215,002	130,826
Investments	5,803,274	5,416,766
Total current assets	6,855,721	6,334,804
Property and equipment, net	137,795	234,343
Total assets	$ 6,993,516	$ 6,569,147
LIABILITIES AND NET ASSETS		
Current liabilities		
Accounts payable and accrued expenses	$ 744,335	$ 77,703
Accrued compensation	59,811	62,459
Deferred revenue	443,275	516,600
Total current liabilities	1,247,421	656,762
Deferred revenue, net of current portion	337,500	487,500
Total liabilities	1,584,921	1,144,262
Net assets - unrestricted		
Undesignated net assets (deficit)	(394,679)	8,119
Board designated	5,803,274	5,416,766
Total net assets	5,408,595	5,424,885
Total liabilities and net assets	$ 6,993,516	$ 6,569,147

USENIX ASSOCIATION

Statements of Activities
Years Ended December 31, 2016 and 2015

	2016	2015
REVENUES		
Conference and workshop revenue	$ 4,857,484	$ 3,679,420
Membership dues	257,295	262,877
Event services and projects	4,750	618,774
Product sales	7,067	6,215
LISA SIG dues and other	7,946	40,482
General sponsorship	60,000	90,000
Total revenues	5,194,542	4,697,768
EXPENSES		
Program services		
Conferences and workshops	4,537,398	3,366,015
Projects, programs and membership	377,969	763,900
LISA SIG	-	3,889
Total program services	4,915,367	4,133,804
Management and general	621,006	595,237
Fundraising	84,715	188,236
Total expenses	5,621,088	4,917,277
CHANGE IN NET ASSETS FROM OPERATIONS	(426,546)	(219,509)
OTHER INCOME (EXPENSES)		
Donations	23,660	26,754
Investment income (loss)	432,343	(30,042)
Investment fees	(45,897)	(49,532)
Other income	150	1,426
Total other income (expenses)	410,256	(51,394)
Change in net assets	(16,290)	(270,903)
NET ASSETS - unrestricted		
Beginning of year	5,424,885	5,695,788
End of year	$ 5,408,595	$ 5,424,885

Chart 1: USENIX 2016 Operating Expenses

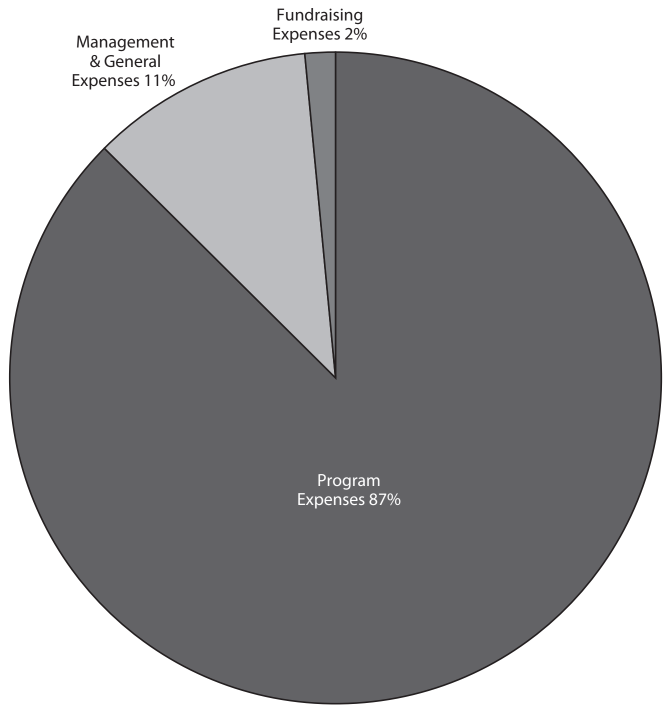


Chart 2: USENIX 2016 Program Expenses

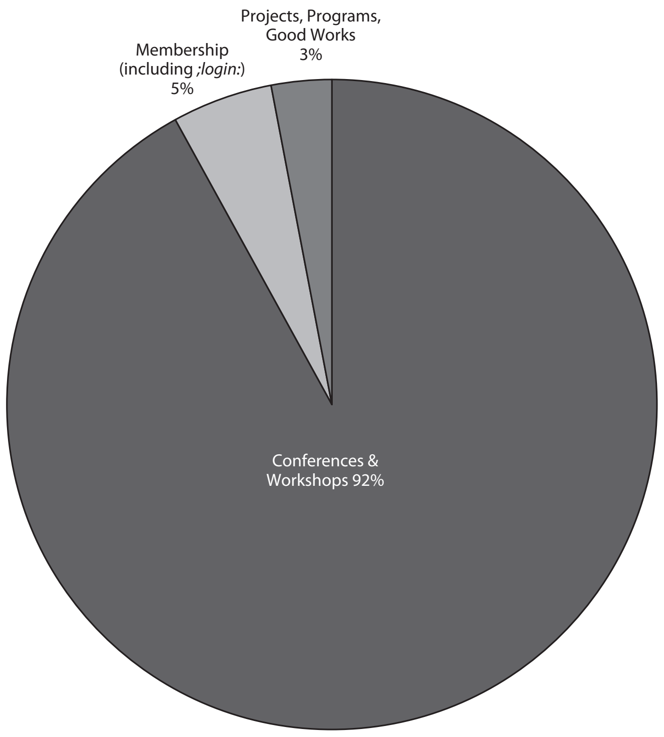
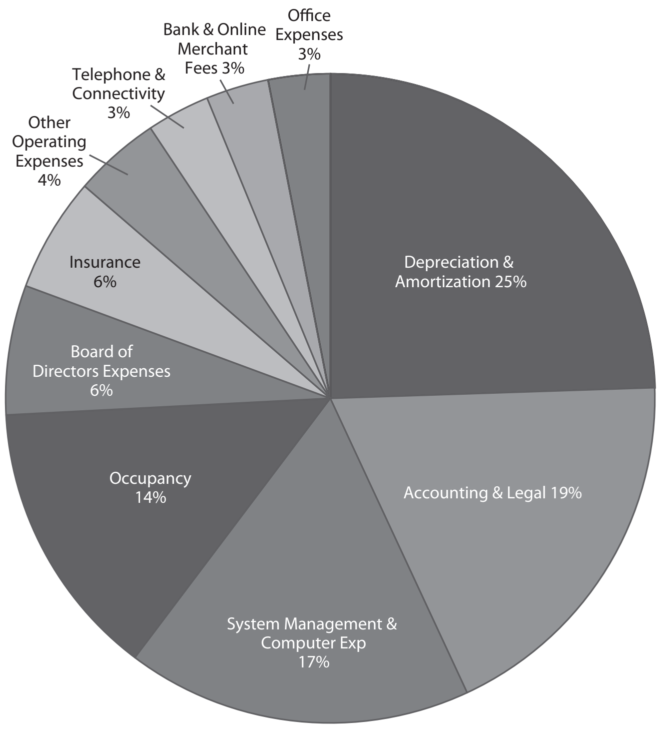


Chart 3: USENIX 2016 General and Administrative Expenses



usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Writing for *;login:*

We are looking for people with personal experience and expertise who want to share their knowledge by writing. USENIX supports many conferences and workshops, and articles about topics related to any of these subject areas (system administration, programming, SRE, file systems, storage, networking, distributed systems, operating systems, and security) are welcome. We will also publish opinion articles that are relevant to the computer sciences research community, as well as the system adminstrator and SRE communities.

Writing is not easy for most of us. Having your writing rejected, for any reason, is no fun at all. The way to get your articles published in *;login:*, with the least effort on your part and on the part of the staff of *;login:*, is to submit a proposal to login@usenix.org.

PROPOSALS

In the world of publishing, writing a proposal is nothing new. If you plan on writing a book, you need to write one chapter, a proposed table of contents, and the proposal itself and send the package to a book publisher. Writing the entire book first is asking for rejection, unless you are a well-known, popular writer.

;login: proposals are not like paper submission abstracts. We are not asking you to write a draft of the article as the proposal, but instead to describe the article you wish to write. There are some elements that you will want to include in any proposal:

- What's the topic of the article?
- What type of article is it (case study, tutorial, editorial, article based on published paper, etc.)?
- Who is the intended audience (syadmins, programmers, security wonks, network admins, etc.)?
- Why does this article need to be read?
- What, if any, non-text elements (illustrations, code, diagrams, etc.) will be included?
- What is the approximate length of the article?

Start out by answering each of those six questions. In answering the question about length, the limit for articles is about 3,000 words, and we avoid publishing articles longer than six pages. We suggest that you try to keep your article between two and five pages, as this matches the attention span of many people.

The answer to the question about why the article needs to be read is the place to wax enthusiastic. We do not want marketing, but your most eloquent explanation of why this article is important to the readership of *;login:*, which is also the membership of USENIX.

UNACCEPTABLE ARTICLES

;login: will not publish certain articles. These include but are not limited to:

- Previously published articles. A piece that has appeared on your own Web server but has not been posted to USENET or slashdot is not considered to have been published.
- Marketing pieces of any type. We don't accept articles about products. "Marketing" does not include being enthusiastic about a new tool or software that you can download for free, and you are encouraged to write case studies of hardware or software that you helped install and configure, as long as you are not affiliated with or paid by the company you are writing about.
- Personal attacks

FORMAT

The initial reading of your article will be done by people using UNIX systems. Later phases involve Macs, but please send us text/plain formatted documents for the proposal. Send proposals to login@usenix.org.

The final version can be text/plain, text/html, text/markdown, LaTeX, or Microsoft Word/Libre Office. Illustrations should be EPS if possible. Raster formats (TIFF, PNG, or JPG) are also acceptable and should be a minimum of 1,200 pixels wide.

DEADLINES

For our publishing deadlines, including the time you can expect to be asked to read proofs of your article, see the online schedule at www.usenix.org/publications/login/publication_schedule.

COPYRIGHT

You own the copyright to your work and grant USENIX first publication rights. USENIX owns the copyright on the collection that is each issue of *;login:*. You have control over who may reprint your text; financial negotiations are a private matter between you and any reprinter.

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

ENIGMA

A USENIX CONFERENCE

SECURITY AND PRIVACY IDEAS THAT MATTER

PROGRAM CO-CHAIRS

Bryan Payne,
Netflix

Franziska Roesner,
University of Washington

The full program and registration are available now.

enigma.usenix.org

JAN 16–18, 2018

SANTA CLARA, CA, USA