

Security

↻ **BeyondCorp: Google Obsoletes the Imaginary Perimeter**

Rory Ward and Betsy Beyer

↻ **Sandboxing with Capsicum**

Pawel Jakub Dawidek and Mariusz Zaborski

↻ **Polypasswordhasher Makes Offline Cracking Unfeasible**

Santiago Torres and Justin Cappos

↻ **Debugging with Failure in Mind**

Peter Gutmann

Columns

Practical Perl Tools: Using CPAN Search Features

David N. Blank-Edelman

Python: Python Constants and Enums

David Beazley

iVoyeur: Re-introducing collectd

Dave Josephsen

For Good Measure: Stress Testing Your Security Posture

Dan Geer

/dev/random: Buying Security Snake Oil

Robert G. Ferrell

Conference Reports

23rd USENIX Security Symposium

SaTCPI '15: National Science Foundation Secure and Trustworthy Cyberspace Principal Investigators' Meeting (2015)

January 5–7, 2015, Arlington, VA, USA

www.usenix.org/satcpi15

FAST '15: 13th USENIX Conference on File and Storage Technologies

February 16–19, 2015, Santa Clara, CA, USA

www.usenix.org/fast15

SREcon15

March 16–17, 2015, Santa Clara, CA, USA

Submissions due January 5, 2015

www.usenix.org/srecon15

NSDI '15: 12th USENIX Symposium on Networked Systems Design and Implementation

May 4–6, 2015, Oakland, CA, USA

www.usenix.org/nsdi15

HotOS XV: 15th Workshop on Hot Topics in Operating Systems

May 18–20, 2015, Kartause Ittingen, Switzerland

Submissions due January 9, 2015

www.usenix.org/hotos15

USENIX ATC '15: USENIX Annual Technical Conference

July 8–10, 2015, Santa Clara, CA, USA

Submissions due February 3, 2015

www.usenix.org/atc15

Co-located with ATC '15 and taking place July 6–7, 2015:

HotCloud '15: 7th USENIX Workshop on Hot Topics in Cloud Computing

Submissions due March 10, 2015

www.usenix.org/hotcloud15

HotStorage '15: 7th USENIX Workshop on Hot Topics in Storage and File Systems

Submissions due March 17, 2015

www.usenix.org/hotstorage15

USENIX Security '15: 24th USENIX Security Symposium

August 12–14, 2015, Washington, D.C., USA

www.usenix.org/usenixsecurity15

Co-located with USENIX Security '15:

WOOT '15: 9th USENIX Workshop on Offensive Technologies

August 10–11, 2015

CSET '15: 8th Workshop on Cyber Security Experimentation and Test

August 10, 2015

FOCI '15: 5th USENIX Workshop on Free and Open Communications on the Internet

August 10, 2015

HealthTech '15: 2015 USENIX Summit on Health Information Technologies

Safety, Security, Privacy, and Interoperability of Health Information Technologies

August 10, 2015

JETS '15: 2015 USENIX Journal of Election Technology and Systems Workshop

(Formerly EVT/WOTE)

August 11, 2015

LISA15

November 8–13, 2015, Washington, D.C., USA

Submissions due April 17, 2015

www.usenix.org/conference/lisa15

Do you know about the USENIX Open Access Policy?

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

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

www.usenix.org/membership

Stay Connected...



twitter.com/usenix



www.usenix.org/facebook



www.usenix.org/youtube



www.usenix.org/linkedin



www.usenix.org/gplus



www.usenix.org/blog

;login:

DECEMBER 2014 VOL. 39, NO. 6

EDITORIAL

2 Musings *Rik Farrow*

SECURITY

- 6 BeyondCorp: A New Approach to Enterprise Security**
Rory Ward and Betsy Beyer
- 12 Sandboxing with Capsicum**
Pawel Jakub Dawidek and Mariusz Zaborski
- 18 PolyPasswordHasher: Improving Password Storage Security**
Santiago Torres and Justin Cappos
- 22 Code Testing through Fault Injection** *Peter Gutmann*
- 26 Capturing Capture the Flag: Further Discussions**
Mark Gondree
- 32 Interview with Dan Farmer** *Rik Farrow*

RESEARCH

- 36 Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications** *Robert Ricci, Eric Eide, and the CloudLab Team*

SYSADMIN

- 39 /var/log/manager: Career Preventative Maintenance Inspections**
Andrew Seely

COLUMNS

- 42 Practical Perl Tools: Oh Say Can You CPAN?**
David N. Blank-Edelman
- 47 All About That Constant** *David Beazley*
- 52 iVoyeur: Rediscovering collectd** *Dave Josephsen*
- 56 For Good Measure: Stress Analysis** *Dan Geer*
- 58 /dev/random: Buying Snake Oil** *Robert G. Ferrell*
- 60 Book Reviews** *Mark Lamourine and Rik Farrow*

USENIX NOTES

- 63 Team USA Brings Home the Gold from IOI 2014**
Brian C. Dean
- 64 Thanks to Our Volunteers**
Casey Henderson
- 66 23rd USENIX Security Symposium**



EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Michele Nelson
michele@usenix.org

COPY EDITORS
Steve Gilmartin
Amber Ankerholz

PRODUCTION
Arnold Gatilao
Jasmine Murcia

TYPESETTER
Star Type
startype@comcast.net

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

www.usenix.org

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

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

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

©2014 USENIX Association
USENIX is a registered trademark of the USENIX Association. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. USENIX acknowledges all trademarks herein. Where those designations appear in this publication and USENIX is aware of a trademark claim, the designations have been printed in caps or initial caps.



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

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

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

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

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

Input Challenged

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

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

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

That means that:

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

was capable of being used to exploit the system.

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

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

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

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

Absurd Answer

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

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

The Lineup

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

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

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

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

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

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

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

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

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

Musings

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

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

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

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

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

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

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

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

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

I don’t think it’s an exaggeration to say that cyber defense solutions will serve as the essential basis for human development and economic growth in this century—I think it’s happening before our very eyes.

—Prime Minister Benjamin Netanyahu [6]

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


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

References

- [1] Gene Spafford quotes: <http://spaf.cerias.purdue.edu/quotes.html>.
- [2] *Lions’ Commentary on UNIX 6th Edition, with Source Code* (Peer to Peer Communications, 1996): http://en.wikipedia.org/wiki/Lions'_Commentary_on_UNIX_6th_Edition,_with_Source_Code.
- [3] LANGSEC: Language-Theoretic Security: <http://www.cs.dartmouth.edu/~sergey/langsec/>.
- [4] “Chomsky hierarchy,” Wikipedia: http://en.wikipedia.org/wiki/Chomsky_hierarchy.
- [5] Patches to Bash: <http://ftp.gnu.org/pub/gnu/bash/bash-4.3-patches/bash43-025>.
- [6] “Netanyahu, Kaspersky, and Gold tackle cyber ‘game-changers,’” EurekAlert! Press Release for Cyber Week 2014: http://www.eurekalert.org/pub_releases/2014-09/afot-cw2092414.php.
- [7] “ISC’s DHCP Client Can Be Used as a Delivery Vector for Bash Bug,” ISC DHCP, <http://www.isc.org/>








Announcing the USENIX Store!



Welcome to the USENIX Store!

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

 <p>Apparel</p>	 <p>Gear</p>	
 <p><i>;login:</i> issues</p>	 <p>Short Topics Books</p>	 <p>Video Box Set USBs</p>

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

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

www.usenix.org/store

BeyondCorp

A New Approach to Enterprise Security

RORY WARD AND BETSY BEYER



Rory Ward is a site reliability engineering manager in Google Ireland. He previously worked in Ireland at Valista, in Silicon Valley at AOL, Netscape, Kiva, and General Magic, and in Los Angeles at Retix. He has a BSc in computer applications from Dublin City University.
roryward@google.com



Betsy Beyer is a technical writer specializing in virtualization software for Google SRE in NYC. She has previously provided documentation for Google Data Center and Hardware Operations teams. Before moving to New York, Betsy was a lecturer in technical writing at Stanford University. She holds degrees from Stanford and Tulane. bbeyer@google.com

Virtually every company today uses firewalls to enforce perimeter security. However, this security model is problematic because, when that perimeter is breached, an attacker has relatively easy access to a company's privileged intranet. As companies adopt mobile and cloud technologies, the perimeter is becoming increasingly difficult to enforce. Google is taking a different approach to network security. We are removing the requirement for a privileged intranet and moving our corporate applications to the Internet.

Since the early days of IT infrastructure, enterprises have used perimeter security to protect and gate access to internal resources. The perimeter security model is often compared to a medieval castle: a fortress with thick walls, surrounded by a moat, with a heavily guarded single point of entry and exit. Anything located outside the wall is considered dangerous, while anything located inside the wall is trusted. Anyone who makes it past the drawbridge has ready access to the resources of the castle.

The perimeter security model works well enough when all employees work exclusively in buildings owned by an enterprise. However, with the advent of a mobile workforce, the surge in the variety of devices used by this workforce, and the growing use of cloud-based services, additional attack vectors have emerged that are stretching the traditional paradigm to the point of redundancy. Key assumptions of this model no longer hold: The perimeter is no longer just the physical location of the enterprise, and what lies inside the perimeter is no longer a blessed and safe place to host personal computing devices and enterprise applications.

While most enterprises assume that the internal network is a safe environment in which to expose corporate applications, Google's experience has proven that this faith is misplaced. Rather, one should assume that an internal network is as fraught with danger as the public Internet and build enterprise applications based upon this assumption.

Google's BeyondCorp initiative is moving to a new model that dispenses with a privileged corporate network. Instead, access depends solely on device and user credentials, regardless of a user's network location—be it an enterprise location, a home network, or a hotel or coffee shop. All access to enterprise resources is fully authenticated, fully authorized, and fully encrypted based upon device state and user credentials. We can enforce fine-grained access to different parts of enterprise resources. As a result, all Google employees can work successfully from any network, and without the need for a traditional VPN connection into the privileged network. The user experience between local and remote access to enterprise resources is effectively identical, apart from potential differences in latency.

The Major Components of BeyondCorp

BeyondCorp consists of many cooperating components to ensure that only appropriately authenticated devices and users are authorized to access the requisite enterprise applications. Each component is described below (see Figure 1).

BeyondCorp: A New Approach to Enterprise Security

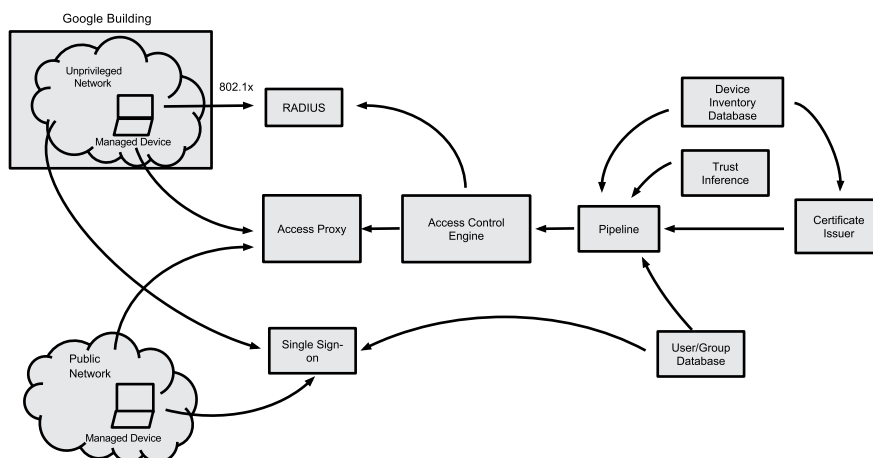


Figure 1: BeyondCorp components and access flow

Securely Identifying the Device

Device Inventory Database

BeyondCorp uses the concept of a “managed device,” which is a device that is procured and actively managed by the enterprise. Only managed devices can access corporate applications. A device tracking and procurement process revolving around a device inventory database is one cornerstone of this model. As a device progresses through its life cycle, Google keeps track of changes made to the device. This information is monitored, analyzed, and made available to other parts of BeyondCorp. Because Google has multiple inventory databases, a meta-inventory database is used to amalgamate and normalize device information from these multiple sources, and to make the information available to downstream components of BeyondCorp. With this meta-inventory in place, we have knowledge of all devices that need to access our enterprise.

Device Identity

All managed devices need to be uniquely identified in a way that references the record in the Device Inventory Database. One way to accomplish this unique identification is to use a device certificate that is specific to each device. To receive a certificate, a device must be both present and correct in the Device Inventory Database. The certificate is stored on a hardware or software Trusted Platform Module (TPM) or a qualified certificate store. A device qualification process validates the effectiveness of the certificate store, and only a device deemed sufficiently secure can be classed as a managed device. These checks are also enforced as certificates are renewed periodically. Once installed, the certificate is used in all communications to enterprise services. While the certificate uniquely identifies the device, it does not single-handedly grant access privileges. Instead, it is used as a key to a set of information regarding the device.

Securely Identifying the User

User and Group Database

BeyondCorp also tracks and manages all users in a User Database and a Group Database. This database system tightly integrates with Google’s HR processes that manage job categorization, usernames, and group memberships for all users. As employees join the company, change roles or responsibilities, or leave the company, these databases are updated. This system informs BeyondCorp of all appropriate information about users that need to access our enterprise.

Single Sign-On System

An externalized, single sign-on (SSO) system is a centralized user authentication portal that validates primary and second-factor credentials for users requesting access to our enterprise resources. After validating against the User Database and Group Database, the SSO system generates short-lived tokens that can be used as part of the authorization process for specific resources.

Removing Trust from the Network

Deployment of an Unprivileged Network

To equate local and remote access, BeyondCorp defines and deploys an unprivileged network that very closely resembles an external network, although within a private address space. The unprivileged network only connects to the Internet, limited infrastructure services (e.g., DNS, DHCP, and NTP), and configuration management systems such as Puppet. All client devices are assigned to this network while physically located in a Google building. There is a strictly managed ACL (Access Control List) between this network and other parts of Google’s network.

BeyondCorp: A New Approach to Enterprise Security

802.1x Authentication on Wired and Wireless Network Access

For both wired and wireless access, Google uses RADIUS servers to assign devices to an appropriate network, based on 802.1x authentication. We use dynamic, rather than static, VLAN assignment. This approach means that rather than relying on the switch/port static configuration, we use the RADIUS servers to inform the switch of the appropriate VLAN assignment for the authenticated device. Managed devices provide their certificate as part of this 802.1x handshake and are assigned to the unprivileged network, while unrecognized and unmanaged devices on the corporate network are assigned to a remediation or guest network.

Externalizing Applications and Workflows Internet-Facing Access Proxy

All enterprise applications at Google are exposed to external and internal clients via an Internet-facing access proxy that enforces encryption between the client and the application. The access proxy is configured for each application and provides common features such as global reachability, load balancing, access control checks, application health checks, and denial-of-service protection. This proxy delegates requests as appropriate to the back-end application after the access control checks (described below) complete.

Public DNS Entries

All of Google's enterprise applications are exposed externally and are registered in public DNS with a CNAME pointing the applications at the Internet-facing access proxy.

Implementing Inventory-Based Access Control Trust Inference for Devices and Users

The level of access given to a single user and/or a single device can change over time. By interrogating multiple data sources, we are able to dynamically infer the level of trust to assign to a device or user. This level of trust can then be used by the Access Control Engine (described below) as part of its decision process. For example, a device that has not been updated with a recent OS patch level might be relegated to a reduced level of trust. A particular class of device, such as a specific model of phone or tablet, might be assigned a particular trust level. A user accessing applications from a new location might be assigned a different trust level. We use both static rules and heuristics to ascertain these levels of trust.

Access Control Engine

An Access Control Engine within the access proxy provides service-level authorization to enterprise applications on a per-request basis. The authorization decision makes assertions about the user, the groups to which the user belongs, the device certificate, and artifacts of the device from the Device Inven-

tory Database. If necessary, the Access Control Engine can also enforce location-based access control. The inferred level of trust in the user and the device is also included in the authorization decision. For example, access to Google's bug tracking system can be restricted to full-time engineers using an engineering device. Access to a finance application can be restricted to full-time and part-time employees in the finance operations group using managed non-engineering devices. The Access Control Engine can also restrict parts of an application in different ways. For example, viewing an entry in our bug tracking system might require less strict access control than updating or searching the same bug tracking system.

Pipeline into the Access Control Engine

The Access Control Engine is constantly fed by a running pipeline that dynamically extracts information useful for access decisions. Among other factors, this information includes certificate whitelists, trust levels of devices and users, and inventory details about the device and the user.

An End-to-End Example

The Application

For this example, let us assume an application is to be taken BeyondCorp. The application is used by engineers to review source code, comment on the code, update the code, and, when approved by reviewers, submit the code. The application, `codereview.corp.google.com`, is restricted to full-time and part-time engineers from any managed device.

Configuring the Internet-Facing Access Proxy

The owner of `codereview.corp.google.com` configures the access proxy for the service. The configuration specifies the location of the back ends and the maximum traffic accepted by each back end. The `codereview.corp.google.com` domain name is registered in public DNS with a CNAME pointing to the access proxy. For example:

```
$ dig @8.8.8.8 codereview.corp.google.com

;<<>> DiG 9.8.1-P1 <<>> @8.8.8.8 codereview.corp.google.com
(1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12976
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0,
ADDITIONAL: 0

;; QUESTION SECTION:
;codereview.corp.google.com. IN A

;; ANSWER SECTION:
codereview.corp.google.com. 21599 IN CNAME
accessproxy.l.google.com.
accessproxy.l.google.com. 299 IN A 74.125.136.129
```

BeyondCorp: A New Approach to Enterprise Security

```

;; Query time: 10 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Aug 20 19:30:06 2014
;; MSG SIZE rcvd: 86

```

Configuring the Access Control Engine

The Access Control Engine provides a default rule that restricts access to full-time employees using a managed device. The owner of `codereview.corp.google.com` provides a more specific rule that further restricts access in two ways: to managed devices with the highest trust level, and to full-time and part-time engineers with the highest trust level.

An Engineer Accesses a Network

If the Network Is Located Outside a Physical Building

Operated by the Enterprise: From a laptop provided by Google, an engineer accesses any WiFi network. For example, this network might be an airport WiFi network with a captive portal or a coffee shop's WiFi. There is no requirement to set up a VPN connection to the enterprise network.

If the Network Is Located in a Physical Building Operated by the Enterprise:

From a laptop or desktop provided by Google, an engineer accesses the enterprise network. The laptop provides its device certificate in the 802.1x handshake with the RADIUS servers. As a valid certificate is provided, the laptop is assigned an address on the unprivileged network. If the device is not a corporate-issued laptop, or its certificate has expired, the device is assigned an address on a remediation network, which has very limited access rights.

Accessing the Application, Regardless of Network

From a corporate-issued laptop on a network, an engineer accesses `codereview.corp.google.com`. You can refer back to Figure 1 as a reference for the flow for this process.

1. The request is directed to the access proxy. The laptop provides its device certificate.
2. The access proxy does not recognize the user and redirects to the SSO system.
3. The engineer provides his or her primary and second-factor authentication credentials, is authenticated by the SSO system, is issued a token, and is redirected back to the access proxy.
4. The access proxy now has the device certificate, which identifies the device, and the SSO token, which identifies the user.
5. The Access Control Engine performs the specific authorization check configured for `codereview.corp.google.com`. This authorization check is made on every request:
 - a. The user is confirmed to be in the engineering group.
 - b. The user is confirmed to possess a sufficient trust level.

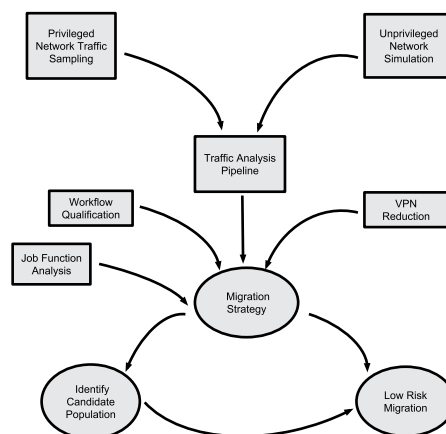


Figure 2: Migrating to BeyondCorp

- c. The device is confirmed to be a managed device in good standing.
- d. The device is confirmed to possess a sufficient trust level.
- e. If all these checks pass, the request is passed to an appropriate back end to be serviced.
- f. If any of the above checks fails, the request is denied.

With this approach, we have rich, service-level authentication and authorization checks that are exercised on a per-request basis.

Migrating to BeyondCorp

Like virtually every other enterprise in the world, Google maintained a privileged network for its clients and applications for many years. This paradigm gave rise to significant infrastructure that is critical to the day-to-day workings of the company. While all components of the company will migrate to BeyondCorp, moving every network user and every application to the BeyondCorp environment in one fell swoop would be incredibly risky to business continuity. For that reason, Google has invested heavily in a phased migration that has successfully moved large groups of network users to BeyondCorp with zero effect on their productivity. The following section, represented by Figure 2, details some of the work we have done.

Workflow Qualification

All the applications used at Google are required to work through the access proxy. The BeyondCorp initiative examined and qualified all applications, which accomplish tasks ranging from the simple (e.g., supporting HTTPS traffic) to the more difficult (e.g., SSO integration). Each application required an access proxy configuration and, in many cases, a specific stanza in the Access Control Engine. Each application went through the following phases:

BeyondCorp: A New Approach to Enterprise Security

1. Available directly from the privileged network and via a VPN connection externally.
2. Available directly from the privileged network and via the access proxy from external and unprivileged networks. In this case, we used split DNS. The internal name server pointed directly at the application, and the external name pointed at the access proxy.
3. Available via the access proxy from external, privileged, and unprivileged networks.

Job Function Analysis

By examining job functions throughout the company and cross-referencing this information against the workflow qualification, we were able to prioritize groups of users to migrate. Therefore, we were able to choose network users from the finance, sales, legal, or engineering groups based upon a thorough understanding of user workflows and the capabilities of the BeyondCorp components at that time.

Cutting Back on the Usage of VPN

As more and more applications became available via the access proxy, we started actively discouraging users from using the VPN, employing the following strategy:

1. We restricted VPN access to users with a proven need.
2. We monitored use of the VPN and removed access rights from users who did not use VPN over a well-defined period.
3. We monitored the VPN usage for active VPN users. If all of their workflows were available through the access proxy, we strongly encouraged users to give up their VPN access rights.

Traffic Analysis Pipeline

It was very important that we moved users to the unprivileged network only when we were certain (or very close to certain) that all of their workflows were available from this network. To establish a relative degree of certainty, we built a Traffic Analysis Pipeline. As input to this pipeline, we captured sampled net-flow data from every switch in the company. This data was then analyzed against the canonical ACL between the unprivileged network and the rest of the company's network. Such analysis allowed us to identify the total traffic that would have passed the ACL, plus an ordered list of traffic that would not have passed the ACL. The non-passing traffic could then be attached to specific workflows and/or specific users and/or specific devices. We then progressively worked through the list of non-passing traffic to make it function in the BeyondCorp environment.

Unprivileged Network Simulation

To augment the Traffic Analysis Pipeline, which used sampled data from switches, we also simulated unprivileged network behavior across the company via a traffic monitor that was installed on all user devices attached to Google's network. The traffic monitor examined all incoming and outgoing traffic on

a per-device basis, validated this traffic against the canonical ACL between the unprivileged network and the rest of the company's network, and logged the traffic that did not pass the validations. The monitor had two modes:

- ◆ Logging mode: captured the ineligible traffic, but still permitted said traffic to leave the device.
- ◆ Enforcement mode: captured and dropped the ineligible traffic.

Migration Strategy

With the Traffic Analysis Pipeline and the unprivileged simulation in place, we defined and are currently implementing a phased migration strategy that entails the following:

1. Identifying potential sets of candidates by job function and/or workflow and/or location.
2. Operating the simulator in logging mode, identifying users and devices that have >99.9% eligible traffic for a contiguous 30-day period.
3. Activating simulator enforcement mode for users and devices that have >99.99% eligible traffic for that period. If necessary, users can revert the simulator to logging mode.
4. After operating the simulator in enforcement mode successfully for 30 days, recording this fact in the device inventory.
5. Along with inclusion in the candidate set, successful operation in the simulator's enforcement mode for 30 days provides a very strong signal that the device should be assigned to the unprivileged network when the next 802.1x authentication request is serviced by the RADIUS servers.

Exemption Handling

In addition to automating the migration of users and devices from our privileged to our new unprivileged network as much as possible, we also implemented a simple process for users to request temporary exemptions from this migration. We maintained a known list of workflows that were not yet qualified for BeyondCorp. Users could search through these workflows, and with the correct approval levels, mark themselves and their devices as active users of a certain workflow. When the workflow was eventually qualified, its users were notified and were again eligible to be selected for migration.

Completing BeyondCorp

The migration of the Google Enterprise to BeyondCorp is well underway, and the majority of workflows it entails are already qualified. Our migration tools and strategy permit us to proactively move users, devices, and workflows to BeyondCorp without affecting day-to-day productivity.

We anticipate a long tail of workflows that will take some time to move to BeyondCorp. For example, fat-client applications that use proprietary protocols to talk to servers will be a challenge.

We are investigating ways to BeyondCorp such applications, perhaps by pairing them with an authentication service.

As we move forward with the migration to BeyondCorp, we intend to publish subsequent articles explaining why and how Google has moved to BeyondCorp, with the goal of encouraging other enterprises in implementing similar strategies.

SAVE THE DATE!

nsdi '15

**12th USENIX Symposium on
Networked Systems
Design and Implementation**

May 4–6, 2015 • Oakland, CA

NSDI '15 will focus on the design principles, implementation, and practical evaluation of networked and distributed systems. Our goal is to bring together researchers from across the networking and systems community to foster a broad approach to addressing overlapping research challenges.

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

www.usenix.org/nsdi15



Sandboxing with Capsicum

PAWEL JAKUB DAWIDEK AND MARIUSZ ZABORSKI



Pawel Jakub Dawidek is a co-founder and CTO at Wheel Systems and a FreeBSD committer who lives and works in Warsaw, Poland. He is the author of various GEOM classes, including the disk-encryption class GELI; he implemented the Highly Available Storage (HAST) daemon for distributing audit trail files (`auditdistd`), and nowadays is mostly working on the Capsicum framework and the Casper daemon.

pjd@freebsd.org



Mariusz Zaborski is currently working as a software developer at Wheel Systems and is a student at Warsaw University of Technology.

He is a successful Google Summer of Code 2013 student. His work is mostly focused on Capsicum and the Casper daemon. Mariusz's relationship with FreeBSD is still young but very intensive. oshogbo@freebsd.org

Very few programmers have managed to successfully use the principle of least privilege, as found in OpenSSH, Postfix, and `djbdns`. Capsicum, introduced in 2010, adds a capability model designed to make it easier for programmers to reason about how to split a program into privileged and unprivileged portions. In this article, we describe the changes made in Capsicum since 2010, compare Capsicum to earlier sandboxing techniques, and look at the new Casperd, which makes it simpler to split programs.

Long ago, people started to recognize that security models proposed by the mainstream operating systems, including Windows, Mac OS X, and all kinds of UNIX-like systems, are simply naive: All you need to do is to write programs that have no bugs. That's indeed naive. Let's also state an obvious rule: The more code we write, the more bugs we introduce, some of which may jeopardize the security of our system. Once we accept this fact, where do we go? We could only develop very small programs, which are easy to audit, but this again would be a bit naive.

To reduce the size of the TCB (trusted computing base), the privilege separation model was introduced. This model splits the program into several independent components, moving all privileged tasks to a small privileged process, and shifting all the work requiring no privileges but that may be risky (like processing network packets) to a larger process that has no privileges. In the case of OpenSSH, the unprivileged process is responsible for parsing all network packets, handling compression, encryption, etc., and the privileged process is responsible for authenticating credentials extracted by the unprivileged process, starting the user's shell, and so on. Those two processes communicate over pipes. Designing the separation properly is very important. If the unprivileged process would have been responsible for authentication and would just pass the result to the privileged process, the whole model would be useless [5, 6].

Global Namespaces

An unprivileged process should be enclosed within some kind of process sandbox. One way to evaluate how good the sandbox is is to check how many global namespaces it is protecting. By global namespace, we are referring to a limited area within the operating system, these areas having some set of names that allow the unambiguous identification of an object [2]. An example of a process sandbox is a Linux kernel mechanism called *seccomp*. This mechanism allows you to limit a process to a state in which you can't do any other system calls than *exit*, *sigreturn*, *read*, and *write* [3]. It appears to be a very secure approach, but it is also very restrictive. For example, you can't, in any situation, open any new file or receive a new file descriptor. Other mechanisms of process sandboxing are Seatbelt (in Mac OS X) and Capsicum (in FreeBSD), which will be described later in this article. In Table 1, we present a full list of the global namespaces in the FreeBSD kernel. One namespace example is the file paths global namespace, which is nothing more than the list of files, symlinks, and directories in our computer.

Namespace	Description
Process ID (PID)	UNIX processes are identified by unique IDs. PIDs are returned by <code>fork</code> and used for signal delivery, debugging, monitoring, and status collection.
File paths	UNIX files exist in a global, hierarchical namespace, which is protected by discretionary and mandatory access control.
NFS file handles	The NFS client and server identify files and directories on the wire using a flat, global file handle namespace. They are also exposed to processes to support the lock manager daemon and optimize local file access.
File system ID	File system IDs supplement paths to mount points, and are used for forcible unmount when there is no valid path to the mount point.
Protocol address	Protocol families use socket addresses to name local and foreign endpoints. These exist in global namespaces, such as IPv4 addresses and ports, or the file system namespace for local domain sockets.
Sysctl MIB	The <code>sysctl</code> management interface uses numbered and named entries, used to get or set system information, such as process lists and tuning parameters.
System V IPC	System V IPC message queues, semaphores, and shared memory segments exist in a flat, global integer namespace.
POSIX IPC	POSIX defines similar semaphore, message queue, and shared memory APIs with an undefined namespace: On some systems, these are mapped into the file system; on others they are simply a flat, global namespace.
System clocks	UNIX systems provide multiple interfaces for querying and manipulating one or more system clocks or timers.
Jails	The management namespace for FreeBSD-based virtualized environments.
CPU sets	A global namespace for affinity policies assigned to processes and threads.
Routing tables	A global namespace with routing tables assigned to process.

Table 1: Global namespaces in the FreeBSD operating system kernel [4]

Table 1 was first published in [4]. Additionally, we would like to include “routing tables” to this list. In the FreeBSD operating system, per-process routing tables may be changed using the program `setfib(1)`.

Security Hacks

In this section, we describe many of the “sandboxing techniques” (or as we prefer, “security hacks”) that were used before process sandboxing, and show that creating an isolated environment wasn’t easy. Programmers try to simulate sandboxes using portable functions like `setuid(2)`, `setrlimit(2)`, `chroot(2)`, etc. Most of these functions are part of the POSIX standard, so they should work on Linux and UNIX operating systems.

setuid(2)*, *setgid(2)*, and *setgroups(2)

It is obvious that unprivileged processes cannot run with root privileges, so they have to run as some other user. In the past, it was common to choose the “nobody” user, but if multiple independent programs reuse this one UID to drop privileges, it may become possible to jump from one program to another. We don’t want that. This is why programs nowadays reserve their own unprivileged users, like the “`sshd`” user in the case of OpenSSH. There are many details you have to do correctly or this won’t work properly:

- ◆ When changing your UID, don’t forget to change your GID, too.
- ◆ When changing your GID, be sure to do it before changing your UID or it will fail.
- ◆ When changing your GID, be sure to remove all the other groups the process owner (root) belongs to, and do it before changing UID.
- ◆ Be sure to use `setgroups(2)`, `setgid(2)`, and `setuid(2)` system calls, or it may be possible to switch back to root.
- ◆ Be sure to verify these operations actually succeed! On some systems, in some conditions, it is not possible for the root user to change its UID, for example, and you’ll be left running as root.
- ◆ Be sure to verify that your target operating system’s `setuid(2)` and `setgid(2)` system calls modify real, effective, and saved user ID and group ID (or use `setresuid(2)/setresgid(2)` if available).
- ◆ Be sure not to modify effective user ID before calling `setuid(2)` or it won’t change saved UID, and it will be possible to switch back to root.
- ◆ Functions that allow you to change UID, GID, and groups require root privileges.

In Listing 1, we have provided an example implementation of this method. It looks easy, doesn’t it? However, there are many examples of people making some slip-up trying to use this technique. The most common mistakes with CVE examples are:

Sandboxing with Capsicum

- ◆ CVE-2013-4559 for `lighttpd`—missing checks for `setuid(2)`, `setgid(2)`, and `setgroups(2)` failures
- ◆ CVE-2007-0536 for `rMake`—missing `setgroups(2)` call
- ◆ CVE-2000-0172 for `mtr`—`seteuid(2)` instead of `setuid(2)`

```
#define VERIFY(expr) do { \
    if (!(expr)) \
        abort(); \
} while (0)
uid_truid, euid, suid;
gid_trgid, egid, sgid;
gid_tgidset[1];
gidset[0] = pw->pw_gid;
if (setgroups(1, gidset) == -1)
    err(1, "Unable to set groups to gid");
if (setgid(pw->pw_gid) == -1)
    err(1, "Unable to set gid");
if (setuid(pw->pw_uid) == -1)
    err(1, "Unable to set uid");
VERIFY(getresuid(&ruid, &euid, &suid) == 0);
VERIFY(ruid == pw->pw_uid);
VERIFY(euid == pw->pw_uid);
VERIFY(suid == pw->pw_uid);
VERIFY(getresgid(&rgid, &egid, &sgid) == 0);
VERIFY(rgid == pw->pw_gid);
VERIFY(egid == pw->pw_gid);
VERIFY(sgid == pw->pw_gid);
VERIFY(getgroups(0, NULL) == 1);
VERIFY(getgroups(1, gidset) == 1);
VERIFY(gidset[0] == pw->pw_gid);
```

Listing 1: Example code to change UID and GID in a secure fashion

Directory Restrictions

The method just described provides us with some security in the file path namespace, but our unprivileged process can still access various files on the system, can fill up file systems like `/tmp/`, or perform network communications. To “fix” the file system problem, we can use the `chroot(2)` system call, which limits access to the file system tree.

Again, a few traps await us here:

- ◆ The `chroot(2)` system call is limited to the root user only, so we need to do it before changing our UID!
- ◆ Once our root directory is changed we have to `chdir(2)` to the new “/” because if the process working directory is outside of the new root directory, it will remain possible to access all the files!
- ◆ Be careful not to leave any directory descriptors open or the process will be able to escape from within our new root directory!

Code which implemented most of these rules is presented in Listing 2. We skipped over checking every open directory descriptor and checking every component for ownership; however, you should be aware that leaving any open directory descriptor is a big mistake.

```
/* Check for open directory descriptors */
/* Check for ownership of every component */
if (chroot(dir) != 0)
    err(1, "Unable to change root directory to \
        %s", dir);
if (chdir("/") != 0)
    err(1, "Unable to change directory
        to new root");
```

Listing 2: Code demonstrating correct use of the `chroot(2)` function

Some examples of common mistakes, with corresponding CVEs:

- ◆ CVE-2008-5110, CVE-2011-4099—missing `chdir("/")` after `chroot(2)`
- ◆ CVE-2005-4532—`chroot` directory writable by user

P_SUGID

After changing our directory using `chroot(2)` and dropping privileges using `setuid(2)`, we are no longer running as root, but all our sandboxes run as the same `UNPRIV_USER` user, which is not good. For example, OpenSSH’s sandbox is using the single `sshd` user to handle sessions from every user that is logging in, including root. Now if we break into such a sandbox we will be running as `sshd` user and can mess with other sandboxes, handling other SSH sessions. What exactly can we do? If we could use `ptrace(2)` to attach to a sandbox that handles root’s session, then we could just modify this sandbox memory and break into root’s SSH session. This possibility alone would make privilege separation useless. Fortunately, this is not possible. Because we were running as root and then dropped our privileges using `setuid(2)`, the kernel tagged our process with the `P_SUGID` flag. On FreeBSD, this prevents another process with the same user ID from being able to debug us. It also means that only some signals may be delivered to such a process, but those signals include `SIGUSR1`, `SIGUSR2`, `SIGHUP`, `SIGALRM`, etc., so it is still not without risk.

As we mentioned in the introduction to this section, most functions presented here are part of the POSIX standard and *should* work on most Linux and UNIX operating systems. Unfortunately, it is not all roses. For example, in 2005, Tavis Ormandy found out that the `setuid(2)` function does not set the `P_SUGID` flag in the NetBSD operating system [9]. So before sandboxing your process using all those techniques, be sure to check that they work properly on your destination operating system.

Very Restrictive Environment

The next thing we shall try to do is to prevent network connections. If an attacker can break into our program, they could, for example, run a spam-sending botnet. One way to prevent network connections is to set the limit on open file descriptors to zero, which will prevent the opening of any new file descriptors and raising the limit back.

Namespace	setuid(1)	chroot(2)	P_SUGID	setrlimit(2)	cap_enter(2)
Process IDs	Unprotected	Unprotected	Partial	Unprotected	Protected
File paths	Partial	Protected	Unprotected	Partial	Protected
NFS file handle	Protected	Unprotected	Unprotected	Unprotected	Protected
Filesystem IDs	Protected	Unprotected	Unprotected	Unprotected	Protected
Sysctl MIB	Partial	Unprotected	Partial	Unprotected	Protected
System V IPC	Unprotected	Unprotected	Unprotected	Unprotected	Protected
POSIX IPC	Partial	Unprotected	Unprotected	Protected	Protected
System clocks	Protected	Unprotected	Unprotected	Unprotected	Protected
Jails	Partial	Unprotected	Unprotected	Unprotected	Protected
CPU sets	Unprotected	Unprotected	Unprotected	Unprotected	Protected
Protocol address	Unprotected	Partial	Unprotected	Protected	Protected
Routing tables	Unprotected	Unprotected	Unprotected	Unprotected	Protected

Table 2: Showing which global namespaces are protected by different sandboxing techniques. Partial means the namespace is protected to some extent.

If we are limiting the number of file descriptors, we could also limit file size and disable forking. If we set the file size limit to zero, a process may not create any new files. Disabling forking will prevent any kind of DDoS attacks that involve running a lot of child processes.

Listing 3 shows example code that sets all of these restrictions.

```
structrlimitrl;
rl.rlim_cur = rl.rlim_max = 0;
if (setrlimit(RLIMIT_NOFILE, &rl) != 0)
    err(1, "Unable to limit file descriptors");
if (setrlimit(RLIMIT_FSIZE, &rl) != 0)
    err(1, "Unable to limit file size");
if (setrlimit(RLIMIT_NPROC, &rl) != 0)
    err(1, "Unable to disallow forking");
```

Listing 3: Example code to create a very restricted environment

This method is used, as far as the authors know, only in OpenSSH. These limits are very restrictive. The process may not receive any new file descriptors, duplicate any descriptors, or open any new files in any situation.

Summary of Security Hacks

These four methods are the most interesting methods to sandbox applications using standard functions. In Table 2, we present information on which method protects which namespace.

While analyzing the Table 2, please keep in mind that using *setrlimit(2)* technique imposes significant restrictions on the programmer and, in the common case, makes *setrlimit(2)* very impractical or even impossible to use.

As you can see, using those techniques leaves a lot of space for mistakes, without even covering all global namespaces. These methods also leave a lot of gaps in global namespaces that they should protect.

Capsicum

Capsicum is a lightweight OS capability and sandbox framework [7]. In FreeBSD, we can divide the architecture of our process sandbox system into two modules:

- ◆ Tight sandboxing (*cap_enter(2)*)
- ◆ Capability rights (*cap_rights_limit(2)*)

By “tight sandboxing” we understand that after calling the *cap_enter(2)* function, the FreeBSD kernel will disallow access to any global namespaces. The kernel will still allow access to any local namespaces, so we can continue to use any references to any part of the global namespace. For example, in the file path namespace you can open a directory (e.g., using the *opendir(2)* function), and after entering the sandbox you can still open any file within that directory (e.g., using the *openat(2)* function).

The second part of Capsicum consists of capability rights, which allow us to limit even more local namespaces. We have a lot of flexibility in setting capability rights, which we can limit to read-only, write-only, or append-only. Many limits are also namespace specific. For example, three file-specific rights are:

- ◆ CAP_FCHMOD allows change mode (*fchmod(2)*).
- ◆ CAP_FSTAT allows getting file stats (*fstat(2)*).
- ◆ CAP_UNLINKAT allows file deletion (*unlinkat(2)*).

Sandboxing with Capsicum

We also have socket-specific rights, for example:

- ◆ `CAP_ACCEPT` accepts connection on socket (*accept(2)*).
- ◆ `CAP_BINDAT` assigns a local protocol address to a socket (*bindat(2)*).

In the FreeBSD operating system, we have defined around 77 capabilities. A full list of the Capsicum capability rights can be found in the FreeBSD `rights(4)` manual page.

Implementation of Capability Rights

In FreeBSD, file descriptors are a carrier of capability rights. In a previous article about Capsicum [1], the authors wrote that we wrap a regular file descriptor structure in a special structure that holds information about rights. That has been changed twice since then. First, they were changed to remove the wrapper structure and add a variable to the `filedesc` structure to describe capability rights. The second modification was to change the type of the variable. Initially, rights were represented by the `uint64_t` type, allowing 64 rights to be defined. It turned out that the maximum number of rights was too small and the `uint64_t` type was changed to a special structure that allows us to define up to 285 rights (and even more if needed with more involved changes).

This new structure is presented in Listing 4. The top two bits in the first element of the `cr_rights` array contain total number of elements in the array plus two. This means if those two bits are equal to 0, we have two array elements. The top two bits in all remaining array elements should be 0. The next five bits in all array elements contain an array index. Only one bit is used and bit position in this five-bit range defines the array index. This means there can be at most five array elements in the future. Using only one bit for array index helps to discover ORing rights from different array elements.

```
#define CAP_RIGHTS_VERSION_00 0
/*
 * #define CAP_RIGHTS_VERSION_01 1
 * #define CAP_RIGHTS_VERSION_02 2
 * #define CAP_RIGHTS_VERSION_03 3
 *
 /
#define CAP_RIGHTS_VERSION CAP_RIGHTS_VERSION_00

struct cap_rights {
    uint64_t cr_rights[CAP_RIGHTS_VERSION + 2]; };
typedef struct cap_rights cap_rights_t;
```

Listing 4: Current structure that defines Capsicum rights

Changing the type of the `cap_rights` structure also forces us to change the interface of the `cap_rights_limit(2)` function. In previous implementations to manage rights, we could simply use logic instructions (e.g., and, or), but now this is no longer possible. New interfaces are presented in Listing 5.

```
/* Interfaces. */
cap_rights_t *cap_rights_init(cap_rights_t *rights, ...);
void cap_rights_set(cap_rights_t *rights, ...);
void cap_rights_clear(cap_rights_t *rights, ...);
bool cap_rights_is_set(const cap_rights_t *rights, ...);
```

Listing 5: New interfaces for managing capability rights

These functions replace the previous logic instructions. First, we need to initialize a `cap_rights_t` structure using `cap_rights_init(2)` function. Then we may add new rights using `cap_rights_set(2)`. Once we finish all the required operation settings, we can use `cap_rights_limit(2)` function to limit a file descriptor. All of these steps are presented in Listing 6.

```
int fd;
cap_rights_t rights;

cap_rights_init(&rights, CAP_READ, CAP_WRITE, CAP_FSTAT);
cap_rights_set(&rights, CAP_FCHMOD);

/* Limit descriptor */
cap_rights_limit(fd, &rights);
```

Listing 6: Example of new interface usage

Status of the Project

Capsicum was first introduced in FreeBSD 9.0 and from then was very quickly developed. Currently, there is ongoing work to port Capsicum sandbox to Linux, OpenBSD, and DragonFlyBSD [7]. A growing list of programs in the FreeBSD operating system now use Capsicum:

- ◆ `auditdistd(8)`
- ◆ `dhclient(8)`
- ◆ `hastd(8)`
- ◆ `hastctl(8)`
- ◆ `kdump(1)`
- ◆ `rwho(1)`
- ◆ `rwhod(8)`
- ◆ `ping(8)`
- ◆ `sshd(8)`
- ◆ `tcpdump(8)`
- ◆ `uniq(1)`

An up-to-date list can be found on the Cambridge Web site about Capsicum in FreeBSD [8].

Casper Daemon

Even though Capsicum gives us more flexibility than other methods, in some cases this is still not enough. Consider the situation in which you need to open a lot of different directories (for example, when sandboxing the `grep(1)` program), or the case where you need to open some Internet connection, but before

entering the sandbox you don't know what kind of connection this will be.

The Capsicum framework resolves this problem using a privilege separation model. Before entering the sandbox you can spawn a new process which will have more access to global namespaces or even may not be sandboxed at all. The privileged process performs some operation like opening files or Internet connections and passes the file descriptor to the unprivileged process using UNIX domain sockets. The unprivileged process performs all other actions. The *rwhod(8)* utility is an example of a program that is sandboxed using this method.

This method works pretty well, but there is a lot of code that would need to be rewritten multiple times for different programs. To solve this problem, the Casper daemon was introduced.

Daemon Architecture

We can separate the Casper daemon into two parts: the Casper daemon itself (*casperd(8)*) and Casper services.

The Casper daemon is a global program in an operating system that waits for connections from other process. We can establish a connection with the daemon using the *cap_init(2)* function. The Casper daemon automatically spawns a second process called the "zygote." The zygote is a light weight process that closes all additional descriptors and uses minimal memory. When a process establishes a connection with the daemon, the process sends information about which services it will require. Casper receives that information and clones the zygote process, and after this operation, one zygote is transformed (using *execv(2)* function) into a service. The process shown in Figure 1 demonstrates these steps.

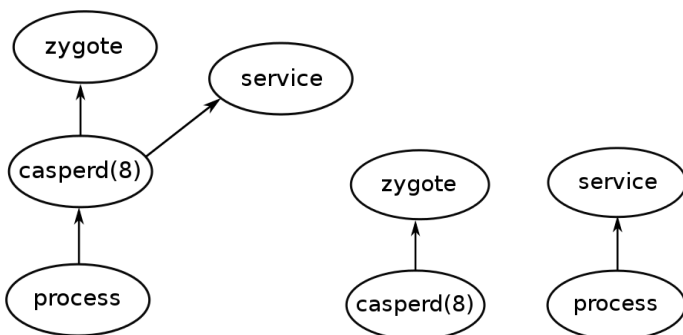


Figure 1: Life cycle of zygote in Casper daemon. On left side, the Casper daemon has spawned a zygote; on the right side, the zygote has been attached to the process-requesting service.

Casper services are specially written programs that have specific tasks. In FreeBSD 11-CURRENT, we have five official services.

- ◆ *system.dns* allows the use of *gethostbyname(3)*, *gethostbyname2(3)*, *gethostbyaddr(3)*, *getaddrinfo(3)*, *getnameinfo(3)*.
- ◆ *system.grp* provides a *getgrent(3)*-compatible API.
- ◆ *system.pwd* provides a *getpwent(3)*-compatible API.
- ◆ *system.random* allows obtaining entropy from */dev/random*.
- ◆ *system.sysctl* provides a *sysctlbyname(3)*-compatible API.

All of these services provide equivalent APIs to the function that they replace.

References

- [1] Robert N. M. Watson, Jonathan Anderson, Ben Laurie, Kris Kennaway, "Introducing Capsicum: Practical Capabilities for UNIX," *login*, vol. 35, no. 6 (December 2010): <https://www.usenix.org/publications/login/december-2010-volume-35-number-6/introducing-capsicum-practical-capabilities-unix>.
- [2] "Namespace," Wikipedia, accessed September 11, 2014: <http://en.wikipedia.org/wiki/Namespcae>.
- [3] Google Seccomp Sandbox for Linux, 2014: <https://code.google.com/p/seccompsandbox/wiki/overview>.
- [4] Robert N. M. Watson, Jonathan Anderson, Ben Laurie, Kris Kennaway, "Capsicum: Practical Capabilities for UNIX," 2010: https://www.usenix.org/legacy/event/sec10/tech/full_papers/Watson.pdf.
- [5] Niels Provos, Markus Friedl, Peter Honeyman, "Preventing Privilege Escalation": <http://niels.xtdnet.nl/papers/privsep.pdf>.
- [6] Niels Provos, Privilege Separated OpenSSH: <http://www.citi.umich.edu/u/provos/ssh/privsep.html>.
- [7] Robert Watson, Cambridge Computer Laboratory Web page, 2014: <https://www.cl.cam.ac.uk/research/security/capsicum/>.
- [8] Robert Watson, Cambridge Computer Laboratory Web page—Capsicum FreeBSD, 2014: <https://www.cl.cam.ac.uk/research/security/capsicum/freebsd.html>.
- [9] Tavis Ormandy, NetBSD Local PTrace Privilege Escalation Vulnerability, CVE-2005-4741: <http://www.securityfocus.com/bid/15290/info>.

PolyPasswordHasher Improving Password Storage Security

SANTIAGO TORRES AND JUSTIN CAPPOS



Justin Cappos is an assistant professor in the Computer Science and Engineering Department at New York University. Justin's research philosophy focuses on improving real world systems, often by addressing issues that arise in practical deployments. jcappos@nyu.edu



Santiago Torres is a graduate student in the Computer Science Department at New York University working under Justin Cappos's mentorship.

He is currently a contributor to open source projects such as "TUF," a secure update framework, and "PolyPasswordHasher," a password storage mechanism resistant to cracking. santiago@nyu.edu

We most often hear about password database thefts and the subsequent cracking of these databases' hashed passwords. Since systems have become faster, and attackers have gained access to clusters or specialized hardware used for cracking, the techniques that have made cracking difficult need to be updated. We have created a system, PolyPasswordHasher, that uses shared keys to add an additional encryption step; it requires an attacker to simultaneously crack several keys at once. We project that PolyPasswordHasher changes the time needed to crack even short passwords to longer than current estimates of the age of the universe.

The Current Standard in Password Protection

Initially, passwords were stored in plaintext on servers. However, once a password database was stolen by an attacker, all passwords on the system could be read. To combat this, password storage systems started to store a cryptographic (one-way) hash of a password. In this scheme, after acquiring a password database, the attacker had to guess at passwords and check their values against the stored hashes in order to recover the actual passwords.

Cracking cryptographic hashes is not as complicated as it sounds, because an attacker can simply pre-compute a database of common passwords and look up a password when given its hash. To address this flaw, "salting" was devised; salt is a random value that is used in the cryptographic hash of the password to make it effectively unique, per database. Current best practice is to create a unique salt for every password (stored alongside the cryptographic hash in the database).

How Do Hackers Steal and Crack Passwords?

To log in, a user provides his or her login name and password to the server. If the user is remote (not physically at the server), this is done over an encrypted channel so that a man-in-the-middle cannot see the user's password. The server receives the user's password, performs a secure salted hash, and checks it against the value stored in the database. If these match, the user is allowed to log in.

When an attacker wants to steal the password for a certain account, there are three options: obtain the password before it gets hashed, act as a man-in-the-middle, or acquire the hash and crack the database. Getting a password before it gets hashed requires the ability to read arbitrary memory (root access) on a running server. Attacks of this nature, in which the server has been completely compromised, account for less than 5% of total compromises, according to Mirante's analysis of recent password hacks [3].

Attacks that try to acquire the password while in transit (as a man-in-the-middle) are even less common. The attacker must both intercept the client's traffic and fool the user into thinking the attacker's site is in fact the actual site they are attempting to log in to. While not perfect, technologies such as SSL and HSTS make thefts that use this technique uncommon.

PolyPasswordHasher: Improving Password Storage Security

The most popular method is for the attacker to obtain a copy of the hashed password database. This commonly occurs when a copy of the hashed password database (e.g., a backup disk) is lost. Attackers also can trigger a hashed password database disclosure, with SQL injections accounting for the majority of known password database breaches.

A hacker who gains access to a hashed password database will usually try to crack passwords on a remote system (offline) by guessing and computing passwords' stored hashes, looking for a match. Cracking programs such as oclHashCat [4] or John the Ripper [2] can automate this process. To give this some perspective, a dump of passwords for 60% of the 6.5 million stolen LinkedIn accounts was found one week after the breach on a hacker forum. This is perhaps not surprising since a security researcher was able to crack 63% of a ~40,000 entry salted SHA1-encoded database in 40 minutes. Given this state-of-affairs, salted password hashes are not a sufficient protection strategy.

A New Defense Scheme: PolyPasswordHasher

To meet the need for enhanced password security, we have created PolyPasswordHasher, a password storage scheme that makes stored password hash data (called polyhashes) interdependent and thus impossible to crack individually. An attacker that obtains a password database stored using PolyPasswordHasher must crack groups of passwords *simultaneously*. The principle that makes this work is the concept of *cryptographic shares*, such as in a Shamir Secret Store [1, 5].

Imagine these cryptographic shares functioning something like a “two-man rule,” such as when a bank check requires multiple signatures or two physical keys must be turned at the same time to open a safety deposit box. A secret key is divided into multiple pieces of information, called shares, with each piece distributed across at least two keyholders. This share strategy aids in the process of recombination. When a certain number of these pieces of information are acquired, an agent is able to recover the original secret key. One important characteristic is that if an agent has only some of the pieces of information needed, they recover no information about the original secret key.

The principal characteristic of this sharing scheme is a configurable *threshold value*, usually set to a value such as 3 or 5, which determines how many shares are needed in order to recover the secret key. The secret key is never stored on disk by PolyPasswordHasher to secure it from attacks such as SQL injection. Instead of storing a secure salted hash, PolyPasswordHasher stores a different value, called a *polyhash*. A polyhash consists of the secure salted hash for the password, XORed with a cryptographic share. This protects a password's secure salted hash with the cryptographic share. That is, before individual passwords can be cracked, an attacker must be able to recover the secret key (recoverable via a threshold of passwords).

Salt	Share Number	PolyHash (Salted Password hash (XOR) Share)
------	--------------	--

Figure 1: How a polyhash is stored for a threshold account

In the following sections, we first describe normal operation of a PolyPasswordHasher server (by assuming that a server has a threshold of passwords, and thus the secret key). We then discuss how a system using PolyPasswordHasher bootstraps after a reboot.

How PolyPasswordHasher Works When a Threshold of Passwords Is Known

PolyPasswordHasher supports two types of user accounts: those that protect a cryptographic share (threshold accounts) and those that do not (thresholdless). Types of accounts that would not protect a share are those in which users are allowed to register any number of accounts, as is the case with Gmail or Facebook. Whether accounts are threshold or thresholdless is invisible to the user, with different procedures taking place in the background.

When a threshold account is created, the system produces a random salt, calculates a salted-hash and issues a new share. The system produces a polyhash by XORing the salted hash and the share, which is then stored, along with the salt and some helper information, as illustrated in Figure 1. The share itself and the salted password hash are never stored on disk.

To log in, a user gives his or her username and password to the server. PolyPasswordHasher checks these to identify which share was assigned to the user's polyhash and then recomputes that share. Next, a salted-hash will be calculated from the input password and its stored salt. Finally, the newly created salted-hash will be XORed with the share to construct a polyhash. Assessing whether the user provided the correct password is a matter of checking the constructed polyhash against the stored polyhash.

If in addition to threshold accounts the system allows other users to freely create accounts (e.g., Gmail), a *thresholdless* entry will be issued for those users. Instead of assigning a share, the secure salted-hash for a thresholdless entry is encrypted with the secret key. Verifying an account for this new user entails decrypting the stored encrypted hash and comparing it in the same fashion as are regular salted-hashes; thresholdless entries are illustrated in Figure 2.

Salt	Encrypted salted Hash
------	-----------------------

Figure 2: Stored data for thresholdless accounts

PolyPasswordHasher: Improving Password Storage Security

Bootstrapping a Server after Reboot

A PolyPasswordHasher server stores its secret key in memory, not on a disk, and the key is thus lost upon reboot. When the server reboots, this secret key is not available, and thus the server cannot compute shares. Therefore, PolyPasswordHasher cannot verify or create accounts as it normally does. PolyPasswordHasher must *bootstrap*.

During this phase, PolyPasswordHasher will collect shares from threshold logins in order to recover the secret. The number of threshold logins required to recover the secret is configured by the system administrator, and it is usually set to a low value (e.g., three or five). For example, if the threshold is three, PolyPasswordHasher will finish bootstrapping after the third threshold account has provided a correct password. While PolyPasswordHasher waits for threshold accounts to log in, it authenticates user passwords using a field called *partial-bytes*.

The partial-bytes field contains only a portion of a regular salted-hash, such as the last four bytes. When a user attempts to log in during the bootstrap phase, PolyPasswordHasher will verify that the partial-bytes field matches the corresponding portion of the password's secure salted hash. For example, if the last four bytes of the salted hash are "A04F," then this will be verified upon login. Although these partial-bytes could *hint* to the attacker what the user's password is, the attacker would not be certain of the password since the complete salted hash is not stored. If the attacker chooses a password that matches the partial-bytes but nonetheless is incorrect, this will be detected after bootstrapping is finished, and the system administrator notified of the likely password hash database theft.

Account creation is also available during the bootstrap phase. To enable this, the new account is added to the database with a regular salted-hash. These accounts can be used normally while the system is bootstrapping. When the system is provided shares from enough threshold accounts, it can finish bootstrapping. To do this, the server re-validates all prior logins with the full polyhash or encrypted salted-hash. Also, any accounts that were created during bootstrap will have their password hash transitioned to protected shares (if threshold) or encrypted shares (if thresholdless).

Evaluation—How We Know It Works

Three elements contribute to the effectiveness of a new password storage method: overhead (e.g., storage and memory costs), efficiency, and time to crack passwords. We assessed storage costs by analyzing the amount of extra information that is required by PolyPasswordHasher and compared that with a standard user database. The only additional information required is the *share number* field and the *partial-bytes* field. The share number requires one extra byte per entry, and the partial-bytes requires four bytes, although this last value is

configurable. The total extra information required is, then, five bytes per entry. Considering that the salt, username, and salted-hash fields account for more than a hundred bytes per entry, we expect the overhead to be less than 5% of the password database storage space cost. Furthermore, the size of a hashed password database is minimal compared to user data (photos, content, etc.) on most systems.

The memory cost of an implementation consists only of a buffer to hold the secret. The size of the buffer for the secret key ranges from 16 bytes to 64 bytes, depending on the implementation.

To understand the instruction efficiency (performance) of PolyPasswordHasher, we performed a series of microbenchmarks on an early 2011 MacBook Pro with 4 GB of RAM and a 2.3 GHz Intel Core i5 processor using a Python reference implementation. We measured instruction efficiency by looking at the time it took for different operations of the PolyPasswordHasher algorithm to complete. We found that the algorithm takes about 150 microseconds to authenticate a user. To transition from the bootstrap phase to normal operation, which is only done once upon restart, takes between hundreds of microseconds to tens of milliseconds after the last threshold account has provided a correct password, depending on the threshold value.

Suppose that users choose passwords from one of the 95 easily typeable characters. If users choose six-character, random passwords, there are only 7.35×10^{11} possible values. When stored with PolyPasswordHasher and a threshold of three, an attacker would need to search 3.97×10^{35} different combinations—more than 23 orders of magnitude more operations.

To put these numbers into perspective, using the best known GPU-cracking techniques, a computer can compute about one billion hashes per second [6]. If three passwords were stored with salted hashes (not PolyPasswordHasher), there are $3 \times 7.35 \times 10^{11}$ combinations possible. It would take an attacker less than an hour to try these combinations on a single computer. With PolyPasswordHasher, to search the keyspace of 3.97×10^{35} combinations would take all 900 million computers on the planet 1.39×10^{10} years. That is longer than the estimated age of the universe.

Summary / What's to Come

There are multiple, open source implementations of PolyPasswordHasher available. Our Django implementation for PolyPasswordHasher is currently being integrated into a variety of servers at New York University. We will use data from these servers to help us understand whether there are any unforeseen complications with production use.

We invite interested parties to find out more information and try out PolyPasswordHasher at: <http://polypasswordhasher.poly.edu>.

PolyPasswordHasher: Improving Password Storage Security

References

[1] K. Hirokuni, "Divide and Manage Secret Data Securely with Shamir's Secret Sharing—Kim's Tech Blog": <http://kimh.github.io/blog/en/security/protect-your-secret-key-with-shamirs-secret-sharing/>.

[2] John the Ripper official Web site: <http://www.openwall.com/john/>.

[3] D. Mirante, J. Cappos, "Understanding Password Database Compromises," Polytechnic Institute of NYU, Department of Computer Science, Technical Report TR-CSE-2013-02 9/13/2013.

[4] oclHashcat official Web site: <http://hashcat.net/oclhashcat/>.

[5] "Shamir's Secret Sharing," Wikipedia: https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing.

[6] A. Zonenberg, "Distributed Hash Cracker: A Cross-Platform GPU-Accelerated Password Recovery System," Rensselaer Polytechnic Institute (2009).

**Help us make another SREcon happen!**

Last May, we held the first ever SREcon, a conference focused in site reliability and production systems at scale. We, the program chairs, wanted to make the event valuable for 200 attendees and capture whether attendees would want to repeat the experience. We viewed SREcon14 as a success because the conference sold out with 275 attendees, and feedback was overwhelmingly positive! Now we need your help to make the next event even better.

The second SREcon will take place on **March 16–17, 2015, in Santa Clara, CA**. We added one more day because we felt that there were many more important subjects to cover than our first program could contain. Now we need to fill in all those spaces, and this is our call for participation. Save the date and come join us for two days of highly technical subjects around site reliability and production at scale.

If you have a talk proposal or panel that is of interest to the community, send us your talk proposal using the template available on the SREcon15 Web site and submit it to srecon15submissions@usenix.org. If you have a suggestion or request for a particular speaker you really would like to see at the conference, feel free to drop us a message, as well. We want SREcon15 to be a high-value conference once more.

Please send us talk proposals until January 5, 2015. We'll evaluate those and get back to you by February 2, 2015.

We are looking forward to seeing you once more!

Program Co-Chairs:
 Sabrina Farmer, Google
 Andrew Fong, Dropbox
 Fernanda Weiden, Facebook

www.usenix.org/srecon15

Code Testing through Fault Injection

PETER GUTMANN



Peter Gutmann is a researcher in the Department of Computer Science at the University of Auckland working on design and analysis of cryptographic security architectures and security usability. He is the author of the open source cryptlib security toolkit, and an upcoming book on security engineering. In his spare time he pokes holes in whatever security systems and mechanisms catch his attention and grumbles about the lack of consideration of human factors in designing security systems.

pgut001@cs.auckland.ac.nz

Several years ago a friend of mine did some robustness testing on a widely used OpenSource Software Library. He instrumented the `malloc()` call so that it would fail (return a NULL pointer/out-of-memory error) the first time that it was called. On the second program run it would fail the second time that it was called, on the next run the third time, and so on. Then he fired up a test suite wrapper for the library and ran it using the fault-inducing `malloc()`.

Luckily, he'd had the foresight to hard-limit the script he was using to stop after a thousand core dumps rather than running through the full test suite wrapper. The hard drive on his computer still hasn't forgiven him for the thrashing it got, though. So why did something as simple as a memory allocation failure cause such havoc?

Why

Most developers have heard that writing unit tests for their code is a Good Thing, and some of them even include the odd one to substantiate this. What these tests invariably do, though, is exercise the standard code paths, the ones that get taken in the presence of normal input and normal operations by other parts of the system. The code paths that handle exception conditions, for example, memory allocation failures, never get tested. It's exactly these conditions that the instrumented `malloc()` exercised, and as the results show, the performance of the never-tested code in these paths was pretty dire.

The instrumented `malloc()` is an example of a testing technique called fault injection, which tests how well (or, more typically, how poorly) code handles exception conditions. The most commonly encountered type of fault injection is fuzz testing or fuzzing, which throws random input at a program to see how it handles it. One of the first instances of fuzz testing looked at the reliability of UNIX utilities in the presence of unexpected input, finding that one-quarter to one-third of all utilities on every UNIX system that the evaluators could get their hands on would crash in the presence of random input [1]. Unfortunately, when the study was repeated five years later the same general level of faults was still evident [2]. While this shows admirable consistency, it's probably not the result that was desired.

Other studies have looked at the behavior of GUI rather than command-line applications in the presence of unexpected input. One such study examined 30 different Windows applications from a mix of commercial and non-commercial vendors. Of the programs tested, 21% crashed and 24% hung when sent random mouse and keyboard input, and every single application crashed or hung when sent random Windows event messages [3]. Before everyone rolls their eyes and mumbles things about Windows, a related study on the reliability of GUI applications under OS X found them to be even worse than the Windows ones [4].

A second type of fault injection involves inducing specific execution failures. One way of doing this is through instrumented system calls like the `malloc()` example given earlier. The late Evi Nemeth of the University of Colorado used to have her students link their programming assignments against a custom `stdlib/libc` in which certain function calls didn't

always succeed unconditionally. Most developers know that you need to check whether a `read()/fread()` actually managed to read the data that it was supposed to, but how many check an `lseek()/fseek()`?

Think of this as a type of control-flow fuzzing rather than the standard data-based fuzzing. Using modified libraries that randomly report failures for system calls that could reasonably be expected to fail, typically implemented as wrappers for standard libraries, makes for a useful testing tool. Some work has already been done in this area, generally looking at ways of automating the creation of fault-injection wrappers for different libraries [5].

The other type of instrumentation that you can use for fault injection is to modify the code itself to inject failures, such as a bit being flipped in digitally signed data, so that the signature check on your SSL handshake fails and your application warns you that the data has been tampered with. If you're thinking "goto fail" or the GnuTLS equivalent at this point, then you'll understand why this type of testing is important.

Implementing this type of fault injection is a bit more laborious than straight fuzzing of either input data or system calls, which rely on the fact that if you make random changes and rerun the code under test often enough then you'll eventually trigger a fault condition.

Statistical fuzzing only works most of the time. If the input data is highly structured, using a tag/length/value or TLV encoding, for example, then any change in the tag or length will be quickly detected and rejected, at least by a properly implemented parser, and any change in the value is irrelevant. To fuzz data like this, you need somewhat exotic and protocol-specific smart fuzzers [6], but that's getting a bit beyond the scope of this article.

What

So if you're trying to catch "goto fail"-style problems, which sorts of faults do you inject and where do you inject them? If what you're implementing conforms to some standard or specification, then the process is, at least in theory, relatively straightforward: You look for any location in the specification where you're required to report an error (e.g., due to a signature check failure) and then inject a fault of that type. If the implementation doesn't detect and report an error, then there's a problem.

The reason why I've said that this works in theory is because most standards seem to focus excessively on the format of messages rather than their semantics. So a standard will describe in minute detail the layout of data elements down to the individual-bit level, but then neglect to mention that if some particular processing step fails you shouldn't continue. For example, here's what the specification for a PKI standard has to say about values that protect against replay attacks:

The [values] protect the PKIMessage against replay attacks. The [value] will typically be 128 bits of (pseudo-) random data generated by the sender, whereas the recipient [value] is copied from the sender [value] of the previous message in the transaction. [7]

That's the entire description (or non-description) of the replay-protection mechanism. Note how the text carefully describes the size of the value and how it's copied around, but never says anything about checking it, or what to do if the check fails. It's possible to create a fully standards-compliant implementation that has no protection whatsoever against replay attacks because the spec never tells you to use the value to defend against these attacks. And if you're relying on using the specification to determine locations for fault injection, you have to infer what you're supposed to do from the comment that the values "protect the PKIMessage against replay attacks."

This problem is widespread among security standards. The OpenPGP specification, which devotes a full 15 pages to the minutiae of the formatting of signatures and signature data, completely omits what exception conditions need to be checked for when processing signatures or how to respond to them. The only comment in the standard that I could find was a statement that "if a subpacket is encountered that is marked critical but is unknown to the evaluating software, the evaluator SHOULD consider the signature to be in error" [8].

The standards that cover SSH are no better, with the sole check that's required being that "values of [Diffie-Hellman parameters] that are not in the range [Diffie-Hellman prime size] MUST NOT be sent or accepted by either side" [9].

As long as an implementation checks those parameters, it can ignore the signature validity check and be completely standards-compliant.

This lack of information unfortunately means that you'll need to go through and annotate the specification to indicate fault conditions that need to be checked at various locations. This can get somewhat tedious, because many specifications are presented more as a catalog of message types (one side sends message A with the following format, the other side responds with message B in the following format, and so on) than a description of the control flow of the protocol.

An additional complication arises because a particular type of failure, and again I'll use the "goto fail" signature-check flaw as the poster child, can have a number of different causes. In this case a signature check could fail because of any corruption/modification of the signed data, incorrect calculation of the hash value that's signed, corruption/modification of the signature value, and incorrect computation of the signature value. So a full-coverage test needs to inject each of these faults in order to verify that the signature-check code will catch all of the different error types.

Code Testing through Fault Injection

When you're thinking about what sorts of faults to inject, you also have to know when to stop. For example, what if you're worried that the code that hashes the data to be signed can detect corruption at the start of the data but not at the end, or a high bit flipped but not a low bit? Eventually, you need to make some assumptions about the correct functioning of standard operations before you start developing an urge to inject faults down at the atomic level.

An alternative strategy that you can use to determine what sorts of faults to apply is to look through the code and make sure that you inject ones that exercise every error path. This isn't such a good approach, though, because it's not certain that the code that you're using as a template to generate your faults is actually checking for all of the error conditions that it's supposed to. This can arise either due to a coding error (the programmer intended to check for an error condition but forgot to add the necessary code, or added the code but got it wrong) or because of a design error (the programmer never even knew that she was supposed to be checking for an error condition). In either case, if the code isn't obviously checking for a fault, you don't know that you should be injecting one.

Figuring out where to inject faults, and what sorts of faults to inject, is by far the hardest part of the process. Once you've done that, you can then get down to implementing the fault injection.

How

Now that you've identified what sorts of faults you want to inject, how do you do it? The most straightforward, but probably also the ugliest, approach is to insert chunks of code inside `#if` blocks that inject faults at appropriate locations. Eventually, you'll end up with the code encrusted in a mass of `#if` blocks controlling conditional compilation, and you'll be hard-pressed to resist taking your former Mona Lisa, now turned into the equivalent of a spray-painted bathroom stall, outside and setting fire to it.

A less inelegant way to handle this is to hide the mess behind a macro, or whatever equivalent your programming language gives you. I use `#define INJECT_FAULT`, taking as argument two parameters, an enum that defines which fault to inject and the code to use to inject the fault. The macro invocation:

```
INJECT_FAULT( FAULT_SIGCHECK, FAULT_SIGCHECK_CODE );
```

expands to:

```
if( faultType == FAULT_SIGCHECK )
{
    fault code defined in FAULT_SIGCHECK_CODE;
}
```

where `faultType` is a global variable that's set to the appropriate fault to inject, and the fault code itself is just a macro-based paste of whatever you need to use to inject the fault. You'll still get a mass of random code to handle the fault injection, but now it's all squirreled away in a header file where you can't see it anymore, or at least where it isn't obviously plastered all over your Mona Lisa.

Finally, you need to exercise your newly added fault-injection capability. This is pretty straightforward: You run your normal test routines, but this time inject one of the faults that you've set up, for example with `setFault(FAULT_SIGCHECK)`. If your test routine still reports success (or if your code simply crashes), then you've got a problem that needs to be addressed. Do this for each fault in turn and make sure that the error is detected.

So that's how you can test your software using fault injection. It won't catch every problem, but it will help you avoid going to fail.

References

- [1] Barton Miller, Lars Fredriksen, and Bryan So, "An Empirical Study of the Reliability of UNIX Utilities," *Communications of the ACM*, vol. 33, no. 12 (December 1990), p. 32.
- [2] Barton Miller, David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, and Jeff Steidl, "Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services," University of Wisconsin-Madison Computer Sciences Technical Report #1268, April 1995.
- [3] Justin Forrester and Barton Miller, "An Empirical Study of the Robustness of Windows NT Applications Using Random Testing," *Proceedings of the 4th USENIX Windows Systems Symposium (WinSys '00)*, August 2000, p. 59.
- [4] Barton Miller, Gregory Cooksey, and Fredrick Moore, "An Empirical Study of the Robustness of MacOS Applications Using Random Testing," *SIGOPS Operating Systems Review*, vol. 41, no. 1 (January 2007), p. 78.
- [5] Paul D. Marinescu and George Candea, "LFI: A Practical and General Library-Level Fault Injector," *Proceedings of the Intl. Conference on Dependable Systems and Networks (DSN)*, June 2009: <http://dslab.epfl.ch/pubs/lfi.pdf>.
- [6] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid and Vitaly Shmatikov, "Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations," *Proceedings of the 2014 Symposium on Security and Privacy (S&P '14)*, May 2014, p. 114.
- [7] Carlisle Adams, Stephen Farrell, Tomi Kause, and Tero Mononen, "Internet X. 509 Public Key Infrastructure Certificate Management Protocol (CMP)," RFC 4210, September 2005.
- [8] Jon Callas, Lutz Donnerhacke, Hal Finney, David Shaw, and Rodney Thayer, "OpenPGP Message Format," RFC 4880, November 2007.
- [9] Tatu Ylonen and Chris Lonvick, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253, January 2006.

Capturing Capture the Flag Further Discussions

MARK GONDREE



Mark Gondree is a security researcher with an interest in games for education and outreach. With Zachary Peterson, he released [d0x3d!], a board game about network security to promote interest and literacy in security topics among young audiences. Gondree is a research professor at the Naval Postgraduate School in Monterey, CA. gondree@gmail.com

Andy Davis is a member of the Cyber Systems Assessment Group at MIT Lincoln Labs. He has helped organize the MIT/LL CTF competition for the last two years and several mini-CTF events at universities in the northeast. Andrew.Davis@ll.mit.edu

Chris Eagle is faculty at the Naval Postgraduate School. He led teams winning DEFCON's CTF competition in 2004 and 2008, and then organized DEFCON's CTF for the next four years, 2009–2012. He is currently designing and organizing DARPA's Cyber Grand Challenge competition. cseagle@nps.edu

Peter Chapman is a graduate student at Carnegie Mellon. He was the first technical lead for picoCTF, an online competition started in 2013 for high school students. He also worked on an attack-defense CTF for the US service academies earlier this year, called IOCTF. peter@cmu.edu

This year, the first USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE) was held, co-located with USENIX Security '14. The summit challenged designers, organizers, gamers, and educators to consider how we assess and improve the current state of security games, both in and out of the classroom.

3GSE featured a panel devoted to capture the flag (CTF) competitions and their use in education, bringing together a diverse group of stakeholders interested in how we both run and evaluate those games. The discussion expressed a fascinating mix of hacker values, student-centric learning approaches, and technical issues inherent to running these complex competitions. I had the opportunity to follow up with our panelists—Peter Chapman, Andrew Davis, Chris Eagle, Portia Pusey, and Giovanni Vigna—to reflect on the highlights of that discussion.

MG: The term “capture-the-flag” has expanded through use. Some might say it has been diluted. Is this confusing? What terminology to distinguish between games seems most useful?

AD: At our CTF, we've had problems with people expecting a type of weekend-long hack-a-thon, where everyone has a project to work on. Some of those types of competitions are advertising themselves as CTFs, so the term is certainly becoming diluted. But we borrowed the term from another game. We've had people show up to our CTF in shorts and a t-shirt, and expect to run around a field stealing flags. So it's partially our own fault.

PC: Within the community, there are some recognized categories but there is a lot of diversity. They'll describe the CTF as: attack-defense, where multiple teams attack each other; *Jeopardy*-style, which is not a great name but refers to challenge-based competitions; and there are war games, which are basically *Jeopardy*-style games that persist, so students can go through challenges and educate themselves at any time. Smash the Stack (<http://smashthestack.org/>) is one such war game. While these categorizations help people know what to expect when they participate, trying to find new terms to recognize “hidden gems” that fall outside these categories will be useful, going forward.

PP: I would add to that list “inherit and defend” competitions. We also need to distinguish CTF from some specialized games such as *Jeopardy*-style, forensics, and cryptography challenges. Working groups at the Cybersecurity Competition Federation (<http://nationalcsf.org/>) have begun to brainstorm and define the diverse competition formats.

CE: What I think is lacking is a categorization that communicates the goals of the organizers. Pure competition play, like DEFCON CTF, appeals to a kind of audience, where the goal is to crown or rank the competitors. But there are other CTFs whose goals are more aligned to education. For these, the value may be in post-exercise debriefings and walkthroughs, which you don't necessarily get from a purely competitive league.

Capturing Capture the Flag: Further Discussions

Portia Pusey is the project manager and education chair for the National Cyber League (NCL). She is part of the research team at the National CyberWatch Center (NCC) who published and presented on the research conducted for the National Cyber League. She is also leading a study of National Collegiate Cyber Defense Competition (CCDC) coaches and players for the NCC Research Team. edrportia@gmail.com

Giovanni Vigna is a professor at UC Santa Barbara, and the director of its Center for CyberSecurity. Since 2003, he has organized the annual International Capture the Flag competition. iCTF is, today, the largest regularly recurring CTF in existence. Play is open to teams from two- and four-year universities. vigna@cs.ucsb.edu

GV: I agree. My gut feeling is that there are many students who would be excited to compete in CTFs but, because we lack advertised goals and expectations, they are scared of participating. Categories that communicate the “roughness” of the game and level of support provided to the players would help. In general, I would call CTF only competitions in which the teams both attack and defend an asset. I would use other terms for other types of competition (such as “hacking competition” for a challenge-based competition).

MG: During the panel, it became clear that some game designers got negative feedback from the community, for failing to be inclusive of professionals or non-US students. What are the limits of inclusivity in CTFs?

PC: When the people running and supporting the game only know English, that becomes a real barrier to supporting schools in some countries.

PP: Each competition can't be everything to everyone; there is a very important reason for that. We need to give young and novice learners a safe legal place to practice. We can't leave them in the “Wild West.” Many high-schoolers that I work with want to build their reputation, and they're getting into trouble. Furthermore, it's not appropriate for minors to interact in the same game environment as adults. And beginners may become disengaged when they have to compete against experts. And, most importantly, games designed to be used in K12 school settings need to protect students' identities and control the types of interactions they have with other players to keep them safe. This comes at the risk of excluding people from games.

MG: How do we build classes around competitions?

CE: For me, I've always found it easier to run a CTF extracurricular activity year-round. Students come and go and may not be able to play year-round, but the timing of these things all over the world is unpredictable and not in harmony with the academic calendar.

AD: We had a professor at a local university use a CTF for their final. I think if you force your students to do a 48-hour, non-stop final where they get beat on, continuously, by more experienced players, that seems pretty cruel.

GV: I disagree with that! I started iCTF in 2001 as an in-class “attacker versus defenders” game, but the defenders claimed it wasn't fun enough. In 2002, we changed the format to “attack-defend,” and in 2003 we opened it up to other universities. It has always been in December, as the “final” for my Fall class. I like the idea of taking a student who knows nothing about security, and building up to running them through a competition, and they enjoy it.

PP: I think it's valuable to separate the idea of competitions that are *educational* and competitions that are designed to be used as *education*. For example, the CCDC is a competition that can frustrate the players to tears. And at the same time, the players say it's the best learning experience they've ever had. So, competitions can be educational. But if a competition is going to be used as formal education there are different requirements. Educators need measurable objectives, and the scoring system needs to provide evidence about the learner's progress towards achieving those objectives. Educators need to know what prerequisite skills are required, and what evidence demonstrates that a learner is ready for the competition challenges. Finally, generally speaking, our nation does not have the capacity among educators in the K12 space to teach cybersecurity. Therefore, we need to provide support in the form of background materials and training for the educators to be able to effectively integrate competitions into their classroom teaching.

Capturing Capture the Flag: Further Discussions

MG: At 3GSE, we heard that some designers are facing pressure to remove scores, make games non-competitive, or turn competition into a series of tutorials. The thinking, in particular, is that competitive play may be unappealing to women. Can we do better outreach by changing design?

PP: If I may speak for all of womankind, we are all different; and some of us are highly competitive. Changing the rules or structure of current competitions is not going to work. It seems to me that evidence from the gaming industry and the current landscape of competitions demonstrates that the games/activities/challenges/tasks do not interest most women. The lesson we learn from the gaming industry is that when a game provides challenges that women want to do, they play. So, the trick to engaging more women in cybersecurity competitions is finding competition tasks that women want to do.

GV: Making CTFs non-competitive is not a good solution to inclusivity. The allure of many CTFs is in their competitive and underground atmosphere. We should seek to draw women to CTFs by finding ways to include more women in computer science. I'm not sure this is a CTF problem but, rather, a larger and more systemic issue facing our field.

CE: We can't expect to have substantially higher percentage of women participating in CTFs than are present in computer science as a whole. The right answer is that we need to address that problem, and then we can start to see participation from women in every aspect of the field and not just CTFs.

PC: There is a difference between building a competition that has been "toned down" to make everyone happy and making a competition where we've removed various barriers to entry. There are ways to build challenges so that they don't hamper the competition but are accommodating to players with less experience. In PicoCTF, for example, some simple challenges have tutorials accompanying them. Experienced players were able to skip the tutorials and solve the challenges quickly. To people with no experience, this was some of their favorite sections. They raved about them, and how they had the support to participate in something they really found interesting. Those teams still didn't score very high in the competition, but they didn't care: they walked away with a very positive experience rather than a feeling that this was too hard for them. We hope to see those players again next year, where they may be able to solve more challenges independently. Removing barriers and letting novices participate in some form is one way to increase our diversity.

PP: Efficacy research among the underrepresented in STEM indicates that, if learning or competition experiences provide a developmental sequence of successes, achievement and interest

in STEM majors and career paths increases. One study documented that a four-week intervention designed to build efficacy helped girls to overcome societal messages and similar pre-existing notions that "women don't or can't do that"; whatever *that* may be. The Cybersecurity Competition Federation (<http://nationalcsf.org/>) is an umbrella organization for competitions; they are building a "pathway" of competitions so that students can identify their point of entry in a continuum of competitions, based on their skill level and interest. It will be interesting to see if this supports greater diversity among players.

MG: If CTFs should be competitive and scored, is there value in tying together team performance across games? Or do we need to measure something in addition to the score?

GV: I think that competitions need to have a ranking, but the value is not in the ranking. The value is in the preparation and the active engagement in the game. One thing we are bad at is evaluating the effort leading up to a competition. We measure the moments in the competition when we are there, but it would be great as educators to find a way to assess their progress.

CE: CTF Time (<https://ctftime.org/>) tries to aggregate information about competitions and weight the games. The organizers have a pretty lofty goal to fill the gap as a team ranking body, but it's hard to do right. CTF teams change and there is no real way to compare competition A to competition B in the absence of a standards body.

PP: The Collegiate Cyber Cup (<http://collegiatecybercup.org/>) is a national award that uses an algorithm to calculate an individual's score based on their performance in multiple competitions (team or individual). It is similar to NASCAR in that points from different competitions are aggregated to identify a national winner. This idea has merit because it provides formative feedback to the player and quantifies performance for future employers. The algorithm is a clever approach to building a national score from the siloed system of competitions we have now. However, I would like to see a common metric that can be used across all competitions based on tasks from a rigorous job performance model such as the one created for the Department of Energy (<https://www.controlsystemroadmap.net/efforts/Pages/SPSP.aspx>).

MG: What factors are most important when designing scoring for a game?

GV: It's a competition and players want to win, so they are very invested in scoring. Most fundamentally, scoring needs to be clear. Most of the times players have been hurt due to scoring, the culprit has been lack of clarity. The scoring rules need to be transparent, and the scoring mechanism needs to be automated, requiring no human or qualitative judgment.

Capturing Capture the Flag: Further Discussions

CE: If you are going to award a prize, your scoring mechanism better be pretty stinking good.

AD: Our CTF group has likely spent more time talking about scoring than anything else, and it's a very hard topic. You have to establish what you want to evaluate. It also needs to be simple enough so that there are no surprises. Anyone should be able to look at the scoring algorithm and see that, if I'm doing what I should be doing in the game and I'm doing better at it than anyone else, then my score should be higher.

CE: When you communicate a scoring mechanism, it should be clear to a player what they should try to optimize to win the game. You may think that good defense could win a game, but when you study the scoring metric then you may find that really offense was what you needed to be doing. Whatever you use as a scoring metric, you better be able to measure it reliably and accurately.

PC: From an educational perspective, we would rather avoid the scenario where someone spends two days trying to hack some binary and doesn't get any points out of it because he couldn't get the last few bits. That's a very frustrating experience and, if we want people to keep learning, we don't want to frustrate them to the point where they stop and leave.

MG: How do we use design or scoring to scaffold challenges, to draw players into developing skills?

PC: In PicoCTF, we had leveled challenges. Our buffer overflow challenges were all discrete problems, but they built on each other, in terms of complexity. There was no downside to the simpler challenges that provided scaffolding: Someone who was very experienced breezed through the simpler challenges and, if anything, it made them feel great for solving five challenges back-to-back. For people who are learning, this disentangles complex problems into more isolated skills.

GV: There may be opportunities for giving partial credit for, say, crashing a program in a predictable way rather than demonstrating a full ROP attack; however, creating alternative goals for partial credit on challenges would needlessly complicate scoring and the game. Rather, the solution for scaffolding is making a variety of simpler challenges.

AD: Each of those smaller challenges will be pretty binary in how they can be scored. Either you've achieved the goal by demonstrating the skill, or you haven't.

MG: It sounds like the types of challenges and the algorithm for scoring communicates a set of values, and has the ability to guide novice players to learn or exercise one set of skills over another. As designers, what do you hope players walk away from the game having achieved or learned?

PC: For PicoCTF, we tried very early to develop a list of skills we hoped to build, but we eventually decided no single set of skills was more valuable than the goal of instilling a curious mindset and a sense of empowerment. We wanted students to question everything, as in the mindset of a computer security expert: You don't trust what people tell you; you don't trust the implementation; instead, you test it and you explore. Additionally, we wanted to empower students to tackle new challenges. Instead of seeing a problem and thinking, "I've never learned this before and that's the teacher's fault and I'm not going to do this anymore," we wanted to instill the sense that everything in the competition will be new to you, and you are going to teach yourself all of it, and you are going to be able to do it. Our game's first challenge was an obscure boot error that essentially required you to find the answer on the Web.

CE: As in teaching my class, my primary goal is demystification: demystify the hardware, demystify the software, remove whatever misconceptions students may have, and empower them to delve more deeply into problems independently.

AD: One thing we did at our CTF, after the competition but before we announced a winner, we had each team spend 30 minutes to make a five-slide presentation. They summarized their offensive strategy, their defensive strategy and gave a rough overview. Each team presented five slides in five minutes; they really got to see, for example, that half the teams had firewall rules that just dropped any packet with five As in row. So, if you wanted your attacks to work, you could have just switched your As to Bs. That reflective period and information sharing has been valuable.

MG: What's next? What is the most pressing need in terms of getting better CTFs?

PP: One of the first problems we need to solve for competitions to be used in education is to make them less time-consuming to create. Once a competition has been played, the solutions are known. All new challenges need to be created for the next game. Until we can solve that problem, it's hard to really tackle problems like scaffolding and scoring.

PC: We thought it would be valuable to release the tools we used to host PicoCTF as an open source project. It has increased the diversity of the competitions. For example, one of the teams playing in PicoCTF took our code and hosted their own nation-wide CTF for high school students called HSCTF (<http://hsctf.com/>). Their twist was to expand the game beyond computer security challenges, to include problems from Project Euler (<https://projecteuler.net/>).

Capturing Capture the Flag: Further Discussions

GV: Ideally, you should be able to just go to a Web site hosting challenges, select the challenges you want, and get VMs you can just spin out to assemble your CTF. In fact, this is exactly what we've recently released in beta, as the iCTF Framework (<https://ictf.cs.ucsb.edu/#/framework>).

AD: One of the problems is that CTF infrastructure developers are not software engineers. Every CTF we've run has incorporated new features that have required pretty experimental software. Our latest CTF heavily employed a new Android emulator that we had just built. There wasn't an existing, mature product to do these experimental CTF challenges. That may be, in part, because there's no real financial support for building and running these games.

CE: Well, there is a market for internal CTFs, where a company will invite an organizer to run a CTF, for training or team building. In the open, however, most people don't receive compensation for running a CTF. It's not pay to play, and the compensation for organizers is rare.

GV: In releasing our framework, our dream is, eventually, to be able to crowd-source the development of vulnerable services, which is the part of design that requires the most human resources. Once you have the infrastructure more or less right, the services are still the things that take time.



Publish and Present Your Work at USENIX Conferences

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

Get more details about each of these Calls for Papers and Participation at www.usenix.org/cfp.

SREcon15

March 16–17, 2015, Santa Clara, CA

Submissions due: January 5, 2015

The second SREcon will take place on March 16–17, 2015, in Santa Clara, CA. Save the date and come join us for two days of highly technical subjects around site reliability and production at scale.

HotOS XV: 15th Workshop on Hot Topics in Operating Systems

May 18–20, 2015, Kartause Ittingen, Switzerland

Submissions due: January 9, 2015

HotOS XV will bring together researchers and practitioners in computer systems, broadly construed. Continuing the HotOS tradition, participants will present and discuss new ideas about systems research and how technological advances and new applications are shaping our computational infrastructure.

2015 USENIX Annual Technical Conference

July 8–10, 2015, Santa Clara, CA

Submissions due: February 3, 2015

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

HotCloud '15: 7th USENIX Workshop on Hot Topics in Cloud Computing

July 6–7, 2015, Santa Clara, CA

Submissions due: March 10, 2015

HotCloud brings together researchers and practitioners from academia and industry working on cloud computing technologies. Cloud computing has gained traction over the past few years, becoming a viable alternative to dedicated data centers and enabling the launch of many prominent companies. However, many challenges remain in the design, implementation, and deployment of cloud computing. HotCloud provides a forum for both academics and practitioners to share their experience, leverage each other's perspectives, and identify new/emerging "hot" trends in this important area.

HotStorage '15: 7th USENIX Workshop on Hot Topics in Storage and File Systems

July 6–7, 2015, Santa Clara, CA

Submissions due: March 17, 2015

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

LISA15

November 8–13, 2015, Washington, D.C.

Submissions due: April 17, 2015

USENIX's Large Installation System Administration (LISA) conference—now in its 29th year—is the premier conference for IT operations, where systems engineers, operations professionals, and academic researchers share real-world knowledge about designing, building, and maintaining the critical systems of our interconnected world. LISA invited submissions of proposals from industry leaders for talks, mini-tutorials, tutorials, panels and workshops. LISA is also interested in research related to the fields of system administration and engineering. We welcome submission for both papers and posters.

Interview with Dan Farmer

RIK FARROW



Dan has been the security architect for four Fortune 500 companies, started his own enterprise software company, and has researched, written, or coauthored a variety of security software tools, papers, essays, and a book (often with his erstwhile colleague Wietse Venema). Someday he'll get that research gig, even if it's after retirement. zen@trouble.org



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

I first met Dan Farmer during DEFCON 1, where I thought he had the most useful and interesting presentation there. I had heard of Dan because he had written COPS, a very early, if not the earliest, vulnerability scanner. I kept encountering Dan over the years at various USENIX conferences as he continued to write tools, papers, and work on improving Internet and *nix security. I also met Wietse Venema for the first time when Wietse and Dan were presenting their forensic toolkit in 1999 [1].

Dan has often appeared in the limelight, partly because he feels so strongly about the general lack of security, but also out of a deep sense of ethics (and contrariness) that has guided his life, often at the expense of his career.

Rik: How did you first get interested in security?

Dan: Growing up I used to love the spy vs. spy mentality in movies and books, but outside of government work there didn't seem to be any way to make a living at it, and I'm afraid I wasn't cut out to live inside the Beltway. But right before I graduated from Purdue there was a massive international security event called the Morris Worm [2]. The network was getting slammed and people were running around in the halls trying to figure out what was going on.

For whatever reason, the Worm captured my imagination like nothing had done before; if computers could do this, there was hope yet. The following semester was due to be my last, but aided by the fact I was a not a good student, I had to take one last course over the summer. Gene Spafford was a professor there and had written one of the two important papers on the Worm, so I simply walked into his office and said I was interested in security and was there any sort of coursework I could do over the summer that would help me graduate.

He agreed to do a special course; I told him I'd like to write something that would test the security of computers, and we worked out the basics of a security tool. It felt like the first time in my life that I had a purpose; I was simply beyond joy to be working on the project. Looking around today it might be hard to imagine that there was just about zero written about security; after months of searching, I had found one book (*UNIX System Security*, by Wood and Kochan), a very small pile of articles, and one jewel, Bob Baldwin's MIT master's thesis that detailed an expert system that probed system security (Kuang). I cobbled together everything I could into one program, which I named COPS, and put it out on the Internet for free.

I thought that that was pretty much it for my security career, but in hindsight my timing was nearly perfect—since there was so little information and zero programs out there, people started assuming I was some sort of security expert rather than an obsessed young lad; and after giving a USENIX paper on COPS [3], I was offered a job at CERT, which was created after the Worm.

It wasn't until many years later that I was able to thank Robert Morris for writing it and starting my career.

Rik: What was it like working for CERT in its early days?

Dan: I think I was the sixth or seventh person hired on at CERT—they had four sharp technical folks already there, and Rich Pethia (a great guy) was leading the charge. Amazingly, a quarter century later, he still manages the group there.

Unless you were around at the time it might be hard to even imagine the paucity of security knowledge then. CERT was a radical idea—set up a 24-hour hotline for anyone in the world to call if they had a security problem, question, or concern. We were one of the only places in the world outside of some tech-savvy governments that knew much of anything about Internet security.

When people started sending us information about break-ins it was a revelation—international intrigue, companies, universities, governments, militaries all over the world getting broken into.

CERT was a good place to work, but I wanted to start researching the latest and greatest, which at the time were network worms and malware. CERT was created by a worm, so why wouldn't they want someone looking at them more in depth? After all, as Sun Tsu reportedly said, to know your enemy you must become them.

Needless to say, they didn't quite agree, but one of my personality defects is my almost pathological contrariness. If people tell me to stay away from something it's something akin to dropping a cardboard box in front of a housecat, we'll both hop right in. So when Sun went looking for a technical head thug in their newly founded security team, I headed for the West Coast; nearly twice the salary didn't hurt either.

Rik: What was it like working in Silicon Valley?

Dan: Silicon Valley was the place to be in the '90s; the Net was exploding, companies zooming up the Fortune 500, and then along came this company called Netscape, and with real money on the line, security started getting the tiniest bit of respect, or at least some modest afterthoughts.

I met Brad Powell in my first stint at Sun in the early '90s; he's a huge guy with a big heart who saw his Biathlon Olympic dreams dashed when Jimmy Carter boycotted the 1980 Olympics because of the Soviet invasion of Afghanistan. Brad wrote a TITAN prototype because he'd run COPS on his customers and was getting really tired of fixing by hand all the myriad security problems COPS would routinely find. We kept in touch after I left, and after haranguing him for years to release his code, he finally agreed if I'd help him spruce TITAN up. TITAN didn't just scan for vulnerabilities, it would also repair them. Brad was a great security guy but at times bore the curse of the engineer and wasn't able to articulate things to mortals.

So I re-architected TITAN, made it possible for normal people (well, normal system administrators!) to actually use the thing, and reassembled and amplified his words to create the USENIX paper [4].

Rik: Where does SATAN fit into the time frame?

Dan: After seeing real incidents at my time at CERT, I became really interested in how people were breaking into computers. At the time security—and especially things like bugs and break-ins and such—were not discussed in polite company, and I couldn't find anyone who had any information at all about how people or programs actually compromised systems. So I sat down and wrote up all the different ways I thought it could happen.

Fortunately, while working at Sun, that fit perfectly into my job, and I had a playground of many thousands of systems that I could legitimately break into. I remember one winter vacation breaking into then-CEO Scott McNealy's workstation and all but one of Sun's 50 VPs' workstations (after asking permission, of course), and only missed that one because the VP had apparently turned off his computer over the break. Fortunately, McNealy was really good-natured when I told him about it in the hallway in passing.

But I didn't feel I had the technical firepower to put out a paper on various ways to break into (and protect) computers on my own, so I reached out to someone I'd never met, a Dutchman by the name of Wietse Venema. Wietse had created the best security tool that had ever been written, TCP Wrapper [5], and seemed perfect for the task—if he wasn't, perhaps he would know who would be. Fortunately, Wietse was intrigued by the project, and we coauthored my favorite project, which detailed how to break into computers along with various defenses you could use to protect yourself.

For some reason we really hit it off and remain close friends to this day. Wietse remains an intellectual with astonishing programming skills, and I was the crazy dreamer who would try to convince him that something utterly ridiculous was a good idea. Postfix, his wonderful mail project and what most people identify him with, was all his idea. Although Wietse dismissed my warnings that it'd take a lot longer than he thought it would and he'd be chaining himself to it for the rest of his life if he went through with it, it all turned out well.

In any case, we mentioned in the paper [6] that we were working on a program called SATAN [7] (Security Analysis Tool for Auditing Networks), never dreaming that people would actually care. However, the paper, the name, and the promise of an automated security scanner struck a chord with our audience, so we started thinking more deeply about how to actually do it.

Interview with Dan Farmer

Perhaps to the disappointment of our audience, we spent the next couple of years writing, talking, and traveling to visit each other; SATAN was perhaps the first security vaporware, the Duke Nukem Forever of its time, until we released it on my birthday in 1995. The delay fueled excitement and anticipation, and the pundits and press had a field day about it all. Fortunately, the Internet survived and it wasn't quite like "randomly mailing automatic rifles" to people or other colorful quotes that found their way into the media. Perhaps the best part of the program was the browser-based UI, which I think was the first of its kind.

Rik: Didn't that result in you losing a job?

Dan: A few months before SATAN's birthday I had gotten a job as the Security Czar of Silicon Graphics (SGI). I still didn't anticipate the fervor to come, but I made sure my boss knew before I was hired of the program and its possibly provocative name. Just prior to the planned release I was called into a meeting, and found myself alone with a vice president and a couple of lawyers, who claimed no prior knowledge of my work or plans for SATAN. I didn't immediately catch on, but soon did after they gave me some options; I could release the program to SGI's customers, I could simply not release it at all, or I could work with SGI to make it a product. Or I could walk.

Another character flaw of mine is saying what I think rather than perhaps being a bit more politic, so I refused their offers to take off with our work and never set foot at SGI again.

Rik: What happened next?

Dan: I went back to work for Sun and was able to do some refueling and research. Along the way I tossed an idea offhandedly to my boss about centralized security management and monitoring; he was pretty stoked about it and asked if Sun could use it for commercial purposes. I replied that I'd have no problem at all with that, but I wasn't going to work on it if it were productized. I had no interest at the time in working in engineering or for a Sun product line.

After helping get a prototype built, we showed it to Eric Schmidt (who later moved on to far greater fame and glory at Google), who gave the order to productize it, and that I'd be the one running the product show. Shortly thereafter I had a conversation with my boss, he said he remembered our deal, but he was ordered to put me in charge...so I quit.

Rik: Jumping ahead to a few years ago, I understand you got some DARPA research money for a security project. Tell us about that.

Dan: My old pal Mudge (Peiter Zatkó) had been running around DARPA for a bit and had helped created something called the Cyber Fast Track program. Mudge had been haranguing me into

submitting a proposal for it. Coincidentally, I'd just been laid off from being Symantec's security architect when they dissolved the entire architecture group as part of a further move towards outsourcing; dozens of us were put out on the street pretty much the same day. Armed with some free time, I submitted a proposal pretty much as a lark; Mudge's claims were so outrageous that it seemed doubtful anything would come of it.

To my surprise, it worked exactly as he claimed. You could submit a small writeup (some two dozen pages at most) about pretty much anything you wanted to work on in security for some months, and within seven working days the US government would say yes or no. I didn't think our government could decide on the time of day in that short a time, let alone grant a contract to work.

Perhaps my favorite work in the Fast Track program was researching IPMI [8], a rather obscure and, as it turns out, insecure out-of-band management protocol that servers speak. The actual project was just a few months of work, but I got intrigued and spent nearly all my free time on it. I ended up working with Fast Track for a couple of years and would be happy to continue similar research, but the program is over now; all things must pass.

Rik: Anything else you'd like to say?

Dan: Over the years, as the tech field has gotten more mature, it seems as though they've squeezed out a lot of the freedom and innovation that fueled the Internet, and more than ever it's simply the financial numbers that matter. Obviously, the numbers do matter, but I don't think it has to be at the expense of everything else. I've no regrets, but if I'd known how it was going to turn out I probably would have gotten that PhD along the way, as the lack of the PhD pretty much excludes me from any institutional research areas, which is where I probably should have gone.

I've been asked many times what they should do by people wanting to get into, or get ahead, in the security business. For me the answer is always the same—follow your heart and give back to the community that helped you get where you are today. This is one of the reasons open source is so important.

For me the hardest thing to do is to keep putting your work and self out there—after all, what the heck do I know compared to all these incredibly smart and capable folks (especially the young ones) who already know computers better than I ever will?

I hope all of this doesn't sound like I'm ungrateful, because I've been extraordinarily fortunate that I've been given the opportunities I've had. I've been called a security expert pretty much the day I got my job at CERT, but I'm pretty dubious about the title—mostly I just had good luck getting into security before most.

References

- [1] The Coroner's Toolkit (TCT): <http://www.porcupine.org/forensics/tct.html>.
- [2] The Morris Worm, a historical view: <http://www.washingtonpost.com/blogs/the-switch/wp/2013/11/01/how-a-grad-student-trying-to-build-the-first-botnet-brought-the-internet-to-its-knees/>.
- [3] D. Farmer, E. Spafford, "The COPS Security System Checker," USENIX Summer Conference, June 11–15, 1990.
- [4] D. Farmer, B. Powell, and M. Archibald, "TITAN," LISA '98: https://www.usenix.org/legacy/publications/library/proceedings/lisa98/full_papers/farmer/farmer.html/farmer.html.
- [5] TCP Wrapper: http://en.wikipedia.org/wiki/TCP_Wrapper.
- [6] D. Farmer and W. Venema, "Improving the Security of Your Site by Breaking into It": <ftp.porcupine.org/pub/security/admin-guide-to-cracking.101.Z>.
- [7] SATAN: http://en.wikipedia.org/wiki/Security_Administrator_Tool_for_Analyzing_Networks.
- [8] Dan Farmer's IPMI research: fish2.com/ipmi/.



Become a USENIX Supporter and Reach Your Target Audience

The USENIX Association welcomes industrial sponsorship and offers custom packages to help you promote your organization, programs, and products to our membership and conference attendees.

Whether you are interested in sales, recruiting top talent, or branding to a highly targeted audience, we offer key outreach for our sponsors. To learn more about becoming a USENIX Supporter, as well as our multiple conference sponsorship packages, please contact sponsorship@usenix.org.

Your support of the USENIX Association furthers our goal of fostering technical excellence and innovation in neutral forums. Sponsorship of USENIX keeps our conferences affordable for all and supports scholarships for students, equal representation of women and minorities in the computing research community, and the development of open source technology.

Learn more at:
www.usenix.org/supporter

Introducing CloudLab

Scientific Infrastructure for Advancing Cloud Architectures and Applications

ROBERT RICCI, ERIC EIDE, AND THE CLOUDLAB TEAM



Robert Ricci is a research assistant professor in the School of Computing at the University of Utah. He is the principal investigator of the CloudLab project and a co-director of the Flux Research Group. He is one of the primary architects and implementers of the Flux Group's many testbeds, including Emulab, ProtoGENI, Apt, and, now, CloudLab.

ricci@cs.utah.edu



Eric Eide is a research assistant professor in the School of Computing at the University of Utah and a co-director of the Flux Research Group.

His research focuses on the design and implementation of trustworthy systems software, including the use of testbeds for repeatable, experimental computer science.

eeide@cs.utah.edu

Additional members of the CloudLab team include Steve Corbató and Jacobus Van der Merwe, University of Utah; Aditya Akella, Remzi H. Arpaci-Dusseau, and Miron Livny, University of Wisconsin-Madison; K.C. Wang, Jim Bottum, James Pepin, and Amy Apon, Clemson University; Chip Elliott and Lawrence H. Landweber, Raytheon BBN Technologies; Michael Zink and David Irwin, University of Massachusetts Amherst; and Glenn Ricart, US Ignite.

Do you have an idea for improving cloud computing? Do you need to instantiate a complete cloud stack so that you can improve part of it? Replace part of it? Tune it to better support a particular scientific workload? This article introduces CloudLab (www.cloudlab.us), a new multi-site facility that we are building to support cloud research.

Researchers and practitioners are flush with ideas for tomorrow's cloud architectures. Their proposals range from small extensions of today's popular cloud-software stacks to all-new architectures that address mobility, energy efficiency, security and privacy, specific workloads, the Internet of Things, and on and on. Many of the ideas that drive modern clouds, such as virtualization, network slicing, and robust distributed storage arose from the research community. However, today's clouds have become unsuitable for moving this research agenda forward: they have specific, unmalleable implementations of the core technologies "baked in."

To support next-generation cloud research, the community needs infrastructure that is built to support research into a wide variety of cloud architectures. CloudLab is a new, large-scale, diverse, and distributed infrastructure designed to address this need. CloudLab is not itself a cloud. Rather, it is a substrate on which researchers can build their own clouds and experiment with them in an environment that provides a high degree of realism.

Like a commercial multi-tenant cloud, CloudLab will be used by many independent experimenters at any given time. In contrast to a commercial cloud, however, CloudLab is being built as a *scientific instrument*. It will give full visibility into every aspect of the facility, and it is being designed to minimize the impact that simultaneous experiments have on each other. This means that researchers using CloudLab will be able to fully understand why their systems behave the way they do, and they can have confidence that the results that they gather are not artifacts of competition for shared hardware resources.

CloudLab is currently under construction by a team located across the University of Utah, Clemson University, the University of Wisconsin-Madison, the University of Massachusetts Amherst, Raytheon BBN Technologies, and US Ignite. Like the team, the CloudLab facility will be geographically distributed, with large clusters at three sites. Each will be a variation on a "reference architecture" that comprises approximately 5,000 cores and 300–500 TB of storage in the latest virtualization-capable hardware. The diversity between sites will help CloudLab to support many areas of research and, at the same time, help researchers test the generality of their findings.

- ◆ The University of Utah site, partnered with HP, will be a cluster with both traditional x86-64 servers and a set of low-power ARM-based servers, enabling researchers to explore power/performance tradeoffs. The cluster will be connected by a large core switch, and it will offer experimenters direct access to switch hardware.
- ◆ The University of Wisconsin-Madison site, partnered with Cisco Systems, will closely reflect the technology and architecture used in modern commercial datacenters. Its 240 servers will have a total of about 4,000 cores and SSDs, and some nodes will have large numbers of disk spindles. They will be connected with a Clos network topology.
- ◆ The Clemson site, developed in cooperation with Dell, will have three components: bulk block-storage nodes, low-density storage nodes for MapReduce/Hadoop-style computing, and generic-VM nodes for provisioning virtual machines. This cluster will focus on provisioning significant experimental environments that can be linked to other national and international resources.

Within each site, CloudLab will provide two 10 Gbps network interfaces to every node. A high-bandwidth switching infrastructure supporting software-defined networking (SDN) will let researchers instantiate a wide range of in-cluster experimental topologies. CloudLab sites will connect with each other via IP and Layer 2 links to regional and national research networks, including AL2S, the SDN-based 100 Gbps network that is part of Internet2's Innovation Platform [7]. This will enable high-speed, end-to-end SDN between all CloudLab sites.

A CloudLab user will be able to provision resources from all of the CloudLab sites at once and combine them into a single experimentation environment. CloudLab's environments will also be able to connect at Layer 2 to the core GENI Network, US Ignite cities [11], and advanced HPC clusters across the United States. For example, in addition to resources from CloudLab's own clusters, a user's environment might include resources from GENI Racks [3], local fiber in a US Ignite city, or cyber-physical systems such as the U. Mass. CASA distributed weather radar system [10].

Like CloudLab's hardware, CloudLab's software is designed for diversity and flexibility in the cloud software stacks that researchers can deploy. CloudLab will be operated by a control framework that runs at a lower layer than cloud software stacks: It will directly provision and control "raw" hardware. A user will request a portion of the raw resources within CloudLab, thereby allocating a *slice* of the CloudLab facility for his or her exclusive use. The ability to allocate resources in this way is familiar to researchers who have used network testbeds such as Emulab [2, 12], GENI [4], DETER [6], and PRObE [8]. In fact, CloudLab's control software will be based on the proven software that today

runs Emulab, several dozen Emulab-based sites, and also parts of GENI.

To allocate a slice of CloudLab, a user writes a *profile*, which is a description of everything needed to build a cloud: both the physical hardware (servers, disks, switches) and the software needed to transform it into a particular type of cloud. (A profile is therefore similar to the definition of an Emulab "experiment" [12] or a GENI "RSpec" [5].) Once the slice is allocated, its owner has full control over its resources. For example, the cloud stack running within the slice can create and manage virtual machines atop the physical machines that are part of the slice. CloudLab will provide canned configurations of popular cloud stacks (e.g., OpenStack [9]), storage systems (e.g., HDFS [1]), and computational frameworks (e.g., Hadoop [1]) so that experimenters can get something running quickly. Researchers will not be bound to these, however. They will be free to deploy whatever they wish on top of the resources provided by CloudLab.

Some researchers will want to create private clouds (e.g., for software development and controlled experimentation), while others will want to open their clouds to other users (e.g., to collect and evaluate real workloads). CloudLab will support both models of experimentation. In addition, researchers will be able to publish their CloudLab profiles, making it straightforward for others to reconstruct the hardware and software environments used in their studies. This will be a mechanism for repeating experiments and comparing results.

We are currently building CloudLab, but we expect that by the time you read this, one or more of the three CloudLab clusters will be up and available to early-access users who can help us to "shake out" the new hardware and software infrastructure. We have an aggressive timetable for building CloudLab, and our plan is that all of CloudLab will be open for regular use in spring 2015. The lessons learned from early adopters will drive the evolution and expansion of CloudLab going forward.

CloudLab will be available without charge to all US academic researchers and educators. In fact, if you have a GENI account or a Utah Emulab account, you can use CloudLab with your existing credentials! We encourage you to try CloudLab if you need modern infrastructure to help you invent the future of cloud computing. Visit the CloudLab Web site (www.cloudlab.us), sign up for news, and email us at contact@cloudlab.us.

This material is based upon work supported by the National Science Foundation under Grant Number 1419199. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Introducing CloudLab

References

- [1] Apache Software Foundation: <http://hadoop.apache.org/>.
- [2] Flux Research Group, University of Utah, Emulab—Network Emulation Testbed Home: <http://www.emulab.net/>.
- [3] GENI Project Office, Raytheon BBN Technologies, Current GENI Rack Projects: <http://groups.geni.net/geni/wiki/GENIRacksHome>.
- [4] GENI Project Office, Raytheon BBN Technologies: <http://www.geni.net/>.
- [5] GENI Project Office, Raytheon BBN Technologies, Resource Specification (RSpec) Documents in GENI: <http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs>.
- [6] Information Sciences Institute, University of Southern California, DeterLab: Cyber-Security Experimentation and Testing Facility: <http://www.deterlab.net/>.
- [7] Internet2, Innovation Platform: <http://www.internet2.edu/vision-initiatives/initiatives/innovation-platform/>.
- [8] New Mexico Consortium, NMC PRObE: <http://www.nmc-probe.org/>.
- [9] OpenStack Foundation, Open Source Software for Building Private and Public Clouds: <http://www.openstack.org/>.
- [10] University of Massachusetts Amherst, Engineering Research Center for Collaborative Adaptive Sensing of the Atmosphere: <http://www.casa.umass.edu/>.
- [11] US Ignite, Ignite Communities: <https://us-ignite.org/hub/>.
- [12] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, December 2002, pp. 255–270.



Do you know about the USENIX Open Access Policy?

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

Please help to us to sustain and grow our open access program. Donate to the USENIX Annual Fund, renew your membership, and ask your colleagues to join or renew today.

www.usenix.org/annual-fund

/var/log/manager

Career Preventative Maintenance Inspections

ANDREW SEELY



Andy Seely is the chief engineer and division manager for an IT enterprise services contract, and an adjunct instructor in the Information and Technology

Department at the University of Tampa. His wife Heather is his PXE Boot and his sons Marek and Ivo are always challenging his thin-provisioning strategy. andy@yankeetown.com

Every good sysadmin knows his systems, understands her role, maintains competency and fluency in current and new technologies, and knows that without a real system administrator on the job the business will ultimately be less effective. Most sysadmins consider all the extra things that go into having a job with a large company to be unpleasant or painful, yet it is attention to the non-technical, the “little things,” that can clear the path for a good sysadmin to be a great contributor to a large team. Not tending to meta-tasks can create a culture of frustration and eventually lead to unemployment. While these scenarios are specific to my own experiences, every business has its own paper cuts.

The Hidden Cost of Not Doing Time Cards

One of my top sysadmins was in the middle of a major product release, working 12-hour days, hurling himself into the implementation of a whole new internal services project. There were maybe two or three people on the whole team who had the knowledge to do this work, spanning network, virtualization, and server specialties, but only one with the drive and experience to carry it through. Everyone knew that the result would be impeccable and the benefits to the business would be huge.

Ours is a contract company where we’re bound to record hours worked every day and to certify the time card every week. Failure to do so has the potential to trigger an external audit and could ultimately result in financial penalties and even loss of contracts [1]. There’s a lot on the line for the company, but the time card system is automated, and for the employee it’s just a two-minute job every day. Maybe three minutes on Friday.

I received an email from my vice president, forwarded to him from the VP of accounting. My top sysadmin had failed to submit his time card. Again. Red flags are popping all the way to the top, and he’s in trouble. Which means that I’m in trouble. I pull him off the product release and have the same conversation. Again. “You’ve got to do your time card.” He gets frustrated. He’s in the middle of an enormous and important task, he’s got his whole mind wrapped around it, and I’m stopping him to talk about his time card. “That’s just stupid,” he snaps at me. “I’m going to fire you, right now, if you don’t do your time card. Right now,” I snap back.

How did we end up in such a surreal, counterproductive, and negative situation? Through lack of common understanding of the whole picture. My sysadmin is focused on the technology and doing a fabulous job of it, but he’s never taken the time to understand his role in the company. It’s partly my fault because I’ve given so many free passes. I’m very susceptible to geniuses doing genius work, and I try to provide as much top cover as possible for the non-technical stuff. My sysadmin isn’t thinking about the danger to our contracts for non-compliance on time reporting. He’s also not realizing that the 11 minutes a week we ask of him to do time reporting turns into several hours of aggregate labor on the part of the payroll, finance, VP, and management teams as they have to prepare reports, excuses, and explanations. He says that he’s there for technology, not for the paperwork, but without the paperwork...he won’t be there at all. The message was received, and he hasn’t missed a time card since.

The Hidden Cost of Lapsed Certifications

My sysadmin responsible for testing new applications in the system integration shop does a great job. His work is excellent, he's well-liked, he's reliable, motivated, and he's got a bright future with the company. He's taking a technical lead role in the development of a new compliance and auditing capability in the enterprise, which the customer expects will result in increased security and flexibility for delivery of all services.

Our contract customer has a policy requiring specific industry certifications for any sysadmin who has elevated rights on the network. The CompTIA Security Plus certification satisfies the minimum requirement. My sysadmin had been certified, but when CompTIA revised their policies and required people certified with Security Plus to register for their continuing education credits program [2], my sysadmin did not register and his certification status lapsed. He felt that it was unfair that CompTIA changed the rules on him, and he had the same knowledge, skills, and abilities today as he did yesterday, so what did it matter?

We perform a regular internal audit of certification status to ensure compliance with customer requirements. Eventually my sysadmin's status was discovered and reported. You never want your name mentioned at the senior leader staff meeting, but at our next meeting there were two names mentioned: his for being noncompliant, and mine for allowing it. Two months of "please," and "it's important," and "just re-certify, it's not difficult" devolved to me having a "your job is on the line" conversation with him. Yes, you do great work. Yes, it's unfair that the rules changed. Yes, you have until next week or we can no longer employ you here because your non-compliance puts our whole contract, not to mention your job and mine, at risk. He took the exam, renewed his certification, and got back to work, but only after his actual job was on the line.

The Hidden Cost of Falling Behind the Technology Curve

My sysadmin was the only expert in a niche technology, a Verity Topic database. The organization was invested heavily in that technology for an enterprise-wide communications platform, and my sysadmin was a wizard; she understood the internals and could fix the most catastrophic database crash barely breaking a sweat. This degree of expertise combined with incredible personal dedication to the company made the sysadmin one of the most important and well-respected members of the team, and it had a secondary effect of allowing the company to save money by keeping the legacy technology in service long past its end of life.

But the technology did pass end of life, and eventually compatibility problems started creeping in. The Verity Topic still ran perfectly, but then a new system couldn't communicate with it and we needed to write some glueware. Then we ran into operating system incompatibilities as we kept the underlying platform updated. The industry moved on and left this niche database technology behind. And left the sysadmin behind, too.

Over the years, she had become comfortable being the one who knew the system, and she made a mistake in thinking that the system would never change. She didn't keep up with advances in other technologies, didn't maintain other relevant certifications, and didn't build any meaningful professional network outside of being well-known as the Verity Topic wizard. When all the technology around the database moved forward, both the database and the sysadmin were left stranded.

We modernized the system and she tried to adapt, but the need to learn something new combined with the pain of no longer being the go-to expert was too much and she left the company and retired from technology work. The company lost an excellent asset and ended up with a reputation with the rest of the team for not "taking care of employees," which had a subtle but real effect on morale and productivity across the organization for months afterwards.

Performing PMI on Careers

All companies and situations have meta-work that has to get done. Ensuring that work gets done and helping employees manage their careers and professional selves is something that a lot of people think is left to the individual employee. When the manager takes a degree of personal responsibility and does "Preventative Maintenance Inspections" on the team, the result can be a smooth and more efficient work force with managed attrition, guided career advancement, and minimum wasted time and effort. I'm the manager, and that's my job.

References

- [1] "Intelligence Agency Billing Fraud Proves Costly," retrieved August 20, 2014: <http://www.washingtontimes.com/news/2014/apr/23/intelligence-agency-billing-fraud-proves-costly/>.
- [2] "CompTIA Certifications Get an Expiration Date," retrieved August 20, 2014: <http://www.gocertify.com/articles/comptia-recertification.html>.



Buy the Box Set!

Whether you had to miss a conference or just didn't make it to all of the sessions, here's your chance to watch (and re-watch) the videos from your favorite USENIX events. Purchase the "Box Set," a USB drive containing the high-resolution videos from the technical sessions. This is perfect for folks on the go or those without consistent Internet access.

Box Sets are available for:

- » **OSDI '14:** 11th USENIX Symposium on Operating Systems Design and Implementation
- » **TRIOS '14:** 2014 Conference on Timely Results in Operating Systems
- » **USENIX Security '14:** 23rd USENIX Security Symposium
- » **3GSE '14:** 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education
- » **FOCI '14:** 4th USENIX Workshop on Free and Open Communications on the Internet
- » **HealthTech '14:** 2014 USENIX Summit on Health Information Technologies
- » **WOOT '14:** 8th USENIX Workshop on Offensive Technologies
- » **URES '14:** 2014 USENIX Release Engineering Summit
- » **USENIX ATC '14:** 2014 USENIX Annual Technical Conference
- » **UCMS '14:** 2014 USENIX Configuration Management Summit
- » **HotStorage '14:** 6th USENIX Workshop on Hot Topics in Storage and File Systems
- » **HotCloud '14:** 6th USENIX Workshop on Hot Topics in Cloud Computing
- » **NSDI '14:** 11th USENIX Symposium on Networked Systems Design and Implementation
- » **FAST '14:** 12th USENIX Conference on File and Storage Technologies
- » **LISA '13:** 27th Large Installation System Administration Conference
- » **USENIX Security '13:** 22nd USENIX Security Symposium
- » **HealthTech '13:** 2013 USENIX Workshop on Health Information Technologies
- » **WOOT '13:** 7th USENIX Workshop on Offensive Technologies
- » **UCMS '13:** 2013 USENIX Configuration Management Summit
- » **HotStorage '13:** 5th USENIX Workshop on Hot Topics in Storage and File Systems
- » **HotCloud '13:** 5th USENIX Workshop on Hot Topics in Cloud Computing
- » **WiAC '13:** 2013 USENIX Women in Advanced Computing Summit
- » **NSDI '13:** 10th USENIX Symposium on Networked Systems Design and Implementation
- » **FAST '13:** 11th USENIX Conference on File and Storage Technologies
- » **LISA '12:** 26th Large Installation System Administration Conference

Learn more at: www.usenix.org/boxsets

Practical Perl Tools

Oh Say Can You CPAN?

DAVID N. BLANK-EDELMAN



David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs. David is honored to have been the recipient of the 2009 SAGE Outstanding Achievement Award and to serve on the USENIX Board of Directors beginning in June of 2010.
dnb@ccs.neu.edu

Every once in a while I get asked to join a conference panel about scripting languages. It will be me, a Pythonista, a Rubyist, and a PHP developer (do they have a cute name?) all onstage together. In most cases, I think the organizers are hoping for the equivalent of a steel cage match in professional wrestling—the fewer participants standing at the end, the better. In these scenarios, I'm almost always a disappointment because I come to praise the other languages, not to bury them. I have a deep appreciation for the other languages, and I'm not afraid to state it even while I'm representing Perl. One of the key reasons I can say “I dig all of the other languages, but I choose to stick with Perl most of the time” is CPAN. This column will focus on CPAN, how to cope with both its triumphs and shortfalls, and some of the ways to interact with it that you may not have encountered before. There probably won't be any code in this issue's column but that's okay because you'll be learning ways to have other people write Perl code for you. We're going to focus on how to consume content from CPAN; discussion about how to contribute to it will have to wait for a future column.

What Is CPAN and How Do I Get Me Some?

I would be really surprised if there are Perl programmers who have never heard of CPAN, but I've been surprised before so pardon me as I go over the basics. CPAN is short for the Comprehensive Perl Archive Network. This is a massive repository of Perl code (largely modules meant for use in other people's code) that has been online since 1995 or so. How massive? As of this writing, cpan.org says:

The Comprehensive Perl Archive Network (CPAN) currently has 138,392 Perl modules in 30,406 distributions, written by 11,739 authors, mirrored on 254 servers.

All of this code has been uploaded so other people may make use of it, so it can be a tremendous resource. Any time you have a problem or a task that sounds like someone else may have solved it, it always behooves you to search CPAN first. We'll talk about ways to do this in a moment.

The plus of having such a massive store of donated code to draw upon is that you often can find someone else has already written (almost) exactly what you need. The minus of having this massive store is some percentage of it is (to be charitable) duplicated effort, and (to be less charitable) some of it is crap. I'll offer tips about this problem later in this column as well.

Deep CPAN Diving

So how do you find what is available on CPAN? Many people start with the search.cpan.org engine. This Googley-looking search engine returns a page like the one in Figure 1.

An experienced CPAN spelunker will scan the returned list of modules and look not just at the description to determine whether a module is appropriate for the task at hand, but also at the metadata. For example, has the module been updated recently? Does it have any reviews

Practical Perl Tools: Oh Say Can You CPAN?

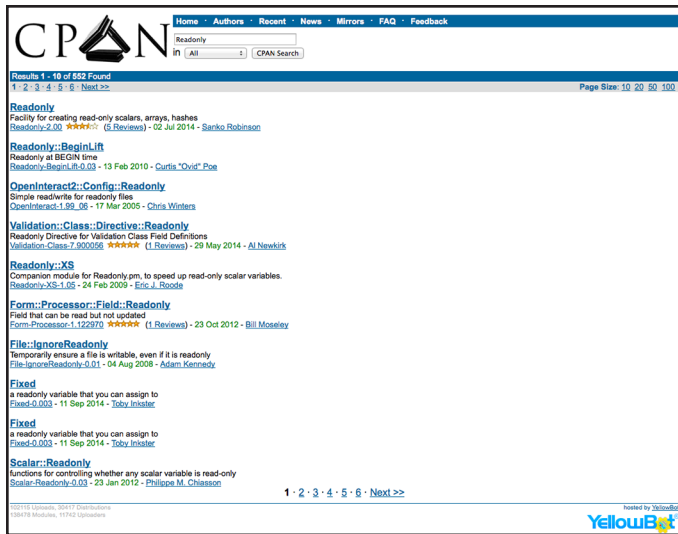


Figure 1: A screenshot of search.cpan.org showing some results

and are they positive? Is the module part of a distribution I recognize? Is the module author well known in the Perl community?, etc.

When you click on the module name, you'll be greeted with the documentation for that module. More often than not, I will click on the breadcrumb link that brings me to the page for the whole distribution (Figure 2).

I do this for two reasons: First, I often want to poke around in a module's code (especially looking at the test code in it for examples of how to use the module). This can be done from the Browse link. Second, I might be curious about bugs filed against the module ("View/Report Bugs") or what other modules this module depends on ("Dependencies"—we'll talk more about that soon). Some of these links can be reached from the search results or the first page linked off the search results, but I'm so used to using the Browse link that going to the distribution page is habitual at this point.

Another way you can search for modules on CPAN is to use metacpan.org. MetaCPAN attempts to be an even spiffier search engine. Figure 3 shows the same search from before, this time run at MetaCPAN.org.

First, let's talk about what is spiffier on the service. When I first started typing "Readonly" into the search box, it attempted to auto-complete my query. Next, not only are Readonly and Readonly::XS next to each other, but at the bottom of the results you can see MetaCPAN has bunched together related modules in a distribution. When I click on the first module link on this page, I see the page that begins like Figure 4.

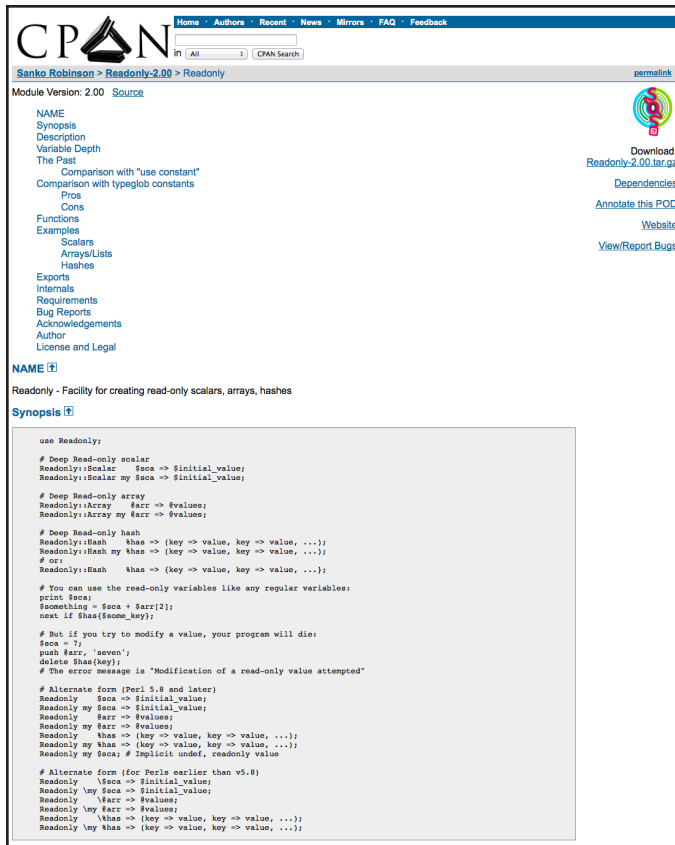


Figure 2: Clicking on the breadcrumb link brings up the page for the whole distribution.

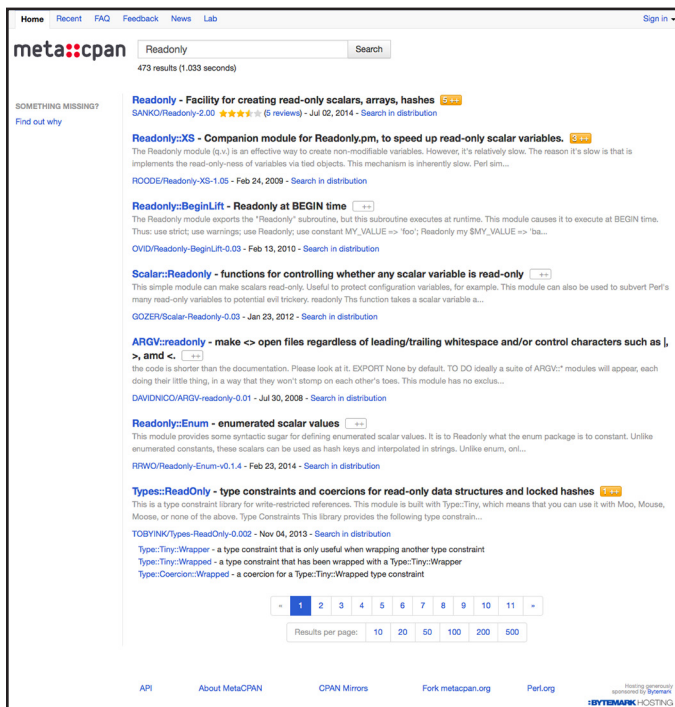


Figure 3: Using MetaCPAN as your search engine

Practical Perl Tools: Oh Say Can You CPAN?

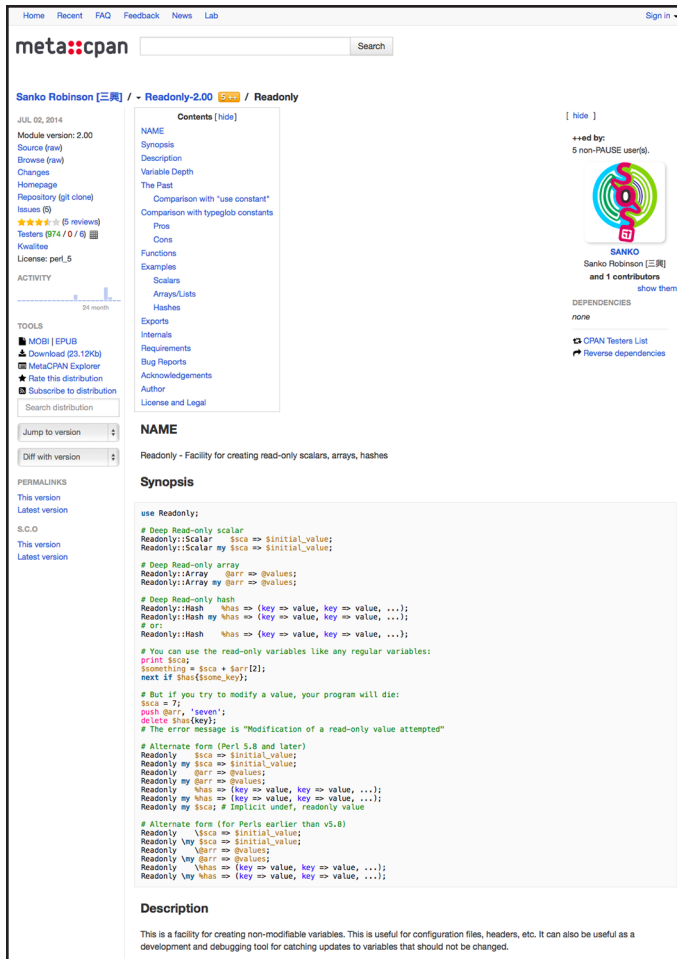


Figure 4: Clicking on the first module link after searching at MetaCPAN

I'd like to call your attention to a few of the things in the left and right sidebars. On the left sidebar, I have the "Browse" link I crave, an easy way to look at the Changelog, pointers to the Git repo, the Bug tracker page for the module, reviews, test results (more on this later), a link to an automated system for determining module quality ("Kwalitee"), an indication how active the development is of the module, and even a way to download the module doc in several ebook formats plus a bunch more stuff. On the right sidebar, we see the dependencies of the module and, perhaps even cooler, a way to see the reverse dependencies (i.e., which modules depend on this one). Super cool.

These are just some of the immediately visible features of MetaCPAN. One thing you probably can't see is a key underlying building block. The search being run on metacpan.org is actually the output of calls to `api.metacpan.org` (documented here: <https://github.com/CPAN-API/cpan-api>). If you'd like to create your own client, the API is open, and example code to use it is freely available. As you can probably guess, metacpan.org is my usual "go-to" search method for searches. I even use this template from within one of my OS X helper applications

(Launchbar) to make looking up module documentation quick and easy: <https://metacpan.org/search?q=>*

Separating the Wheat from the Chaff

Now that you know how to find more modules than you can shake a stick at, how do you figure out which are the good ones? I've mentioned a couple of ideas already in passing, but let's take a closer look at this question.

First, I think it is worthwhile to favor modules that appear to be actively maintained. This increases the likelihood that there is an author out there improving the module and also available to fix issues should you find any. The last release date is a good hint about this, the activity indicator provided by metacpan.org is an even better indication.

Second, closely related to the first idea is the number of bugs opened against the module. I don't believe zero active bugs is necessarily a good thing. I'd much rather see a few open bugs (shows community involvement) alongside a number of closed bugs (shows author involvement and responsiveness). A queue full of unresolved bugs is also a great red flag that may indicate an orphaned module. Use this as one of your parameters for judgment but not the only one.

Third, consider whether the module appears to actually work. One way to determine this is to look at the Testers link off of the metacpan module page for a module. To return to the panels I mentioned at the beginning of the column, another thing I believe Perl can be proud about is the strong cultural inclination towards testing in the community. One way this manifests is that every version of every module that gets submitted to CPAN gets "smoke tested" on close to a thousand different combinations of Perl versions and operating systems. If a module includes tests (and indeed, every module is encouraged to have as complete a test suite as possible), these tests are run in each of these environments and the results reported back to the central test result repository for you to peruse. This gives you a good indication (again, if the test suite is decent) of how portable and how fragile the module code is likely to be. I've mentioned a couple of times that a good test suite makes this metric useful, and I'd recommend using the Browse link to see what sort of tests are included with a module. Similarly, if you browse and find the module has a README that contains boilerplate that the author hasn't bothered to change along the lines of:

The README is used to introduce the module and provide instructions on how to install the module, any machine dependencies it may have (for example, C compilers and installed libraries) and any other information that should be provided before the module is installed....

(which is how the boilerplate provided by `Module::Starter` begins), that's generally a bad sign. There are other signs of slapdashery

Practical Perl Tools: Oh Say Can You CPAN?

you can find in module land, but this is one of my favorite indicators.

Fourth, pay attention to both the dependency and reverse-dependency hints provided by `metacpan.org`. Looking at the dependency list for a module can give you some sense of things such as how hard it will be to get a module to install (do you already have the dependencies installed?), does the author tend to use already existing code (which can be good or bad) or prefer to rewrite everything from scratch, and in general what modules that author trusts.

I find the list of reverse dependencies to sometimes be an even more useful metric for module trustworthiness. Using the same basic underlying principle of Google PageRank, if lots of other modules depend on a module you are considering installing, that's almost always a really good thing. If there's a problem with the module, a whole bunch of other module authors have an incentive to see that problem resolved. Similarly, those authors are also invested in the continued stability and incremental improvement of the module you are considering. In case you are curious, according to the CPAN Top 100 site (<http://ali.as/top100/>), the module with the most dependencies is `App::Munchies` (a Catalyst demonstration Web app), and the module that the most other modules depend on is `Test::Harness`.

Fifth, and my last tip for picking good modules, is to find an opinionated author/expert you trust and follow their advice. Two examples of this are Damian Conway's *Perl Best Practices* (full disclosure, published by the same publisher as my book) and the `Task::Kensho` module. This module is basically a list of recommended modules, or as their doc puts it:

Task::Kensho is a list of recommended modules for Enlightened Perl development. CPAN is wonderful, but there are too many wheels and you have to pick and choose amongst the various competing technologies.

The list looks very solid to me, so I think you can't go wrong at least consulting it as part of your decision process.

Gimme, Gimme

Now that you've found the module of your dreams, how do you go about using it? There's a decision tree here that many a sysadmin has argued about in the past, namely do you install modules using the language native method or do you strictly only use pre-built packages in the context of the package management system your operating system uses (even if you have to build the package yourself). Let's look at both roads.

Back in the early days, people used a module called `CPAN.pm` to install their Perl modules. Later on, a spiffier version was created called `CPANPLUS`, and that's a fairly common way to install modules. It installs a command-line program called "cpanp" that you can run and use like this:

```
$ cpanp
CPANPLUS::Shell::Default -- CPAN exploration and module
installation (v0.9121)
*** Please report bugs to <bug-cpanplus@rt.cpan.org>.
*** Using CPANPLUS::Backend v0.9121. ReadLine support
enabled.

*** Type 'p' now to show start up log

CPAN Terminal>i Readonly
```

This will search for and install the `Readonly` module (and all of the dependencies it has). By default it is fairly interactive, asking you each step of the way whether you want to install each dependency. This isn't my current method for module installation, but before I move on to what I prefer, let me mention one thing `CPANPLUS` does that is valuable. Instead of using the "i" command for install, typing "o" will output a list of the outdated modules on your system. Sort of like this:

```
1  1.5701  1.61  App::Cpan      BDFOY
2  0.58    0.68  Archive::Extract BINGOS
3  1.82    1.90  Archive::Tar    BINGOS
4  5.72    5.73  AutoLoader     SMUELLER
5  1.17    1.18  B::Debug       RURBAN
6  1.14    1.17  B::Lint        RJBS
7  1.52    1.59  CAM::PDF       CDOLAN
8  3.59    3.63  CGI            MARKSTOS
...
```

The second column is the version you have installed, the third is the latest version found on CPAN. This can be very handy if you like to keep current.

My use of `CPAN.pm` and `CPANPLUS` has almost entirely been supplanted by a package called `CPANMINUS`. I typically use it in conjunction with the `perlbrew` system (<http://perlbrew.pl>), which allows you to have multiple versions of Perl installed on your system without conflict (including conflict with the one that ships with your operating system). If you are using `perlbrew`, "perlbrew install-cpanm" will install it for you. If you are not using `perlbrew`, there are a number of ways to install it, including this scary, scary way:

```
$ curl -L http://cpanmin.us | perl - App::cpanminus
```

See <http://cpanmin.us> for more details.

Once you have `CPANMINUS` installed, you will have a "cpanm" command. "cpanm" can take a few flags to modify its behavior, but more often than not, you'll just be typing:

```
$ cpanm {module name}

as in

$ cpanm Readonly
```

Practical Perl Tools: Oh Say Can You CPAN?

CPANMINUS will install the module and any dependencies it has lickety-split with basically no interaction and no fuss. It is really written for the “I want the thing. Thing is now installed.” experience and does it very well.

So that’s how you would install things independent of any package management system your operating system uses. Some find this to be fine; others feel it is reckless and contrary to the reason one has a package management system. If you want to stick to a package system, I do know that in addition to package manager-specific tools like `dh-make-perl`, the awesome FPM tool (<https://github.com/jordansissel/fpm>) by Jordan Sissel can also help create packages for you.

So, with that, we’ve learned how to find good Perl modules and install them easily. Let’s leave it there. Take care and I’ll see you next time.

All About That Constant

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

I'm not sure I've ever seen "gravitas" used as a kind of software metric. However, if there were such a thing, I think it could probably be measured by the number of predefined constants required to carry out any sort of task. For example, directly programming with sockets and setting socket options has a high degree of gravitas. Simply specifying a port number to a Web framework—not so much. Other examples might include programming OpenGL vs. turtle graphics. Or maybe just about anything involving OpenSSL. Extra bonus points are earned if such constants can get together in an unholy bitmask such as `O_RDWR | O_CREAT`. Yes, constants. Gravitas.

Constants, or shall I say "constants," have always been relatively easy to define in Python. Simply create some variables:

```
AF_UNIX = 1
AF_INET = 2
AF_IPX = 23
AF_INET6 = 30

SOCK_STREAM = 1
SOCK_DGRAM = 2
SOCK_RAW = 3
```

Then, pass these values along whenever you need to use them:

```
sock = socket(AF_INET, SOCK_STREAM)
```

Yes, it's pretty simple stuff. Of course, all of those constants are really just simple variables. And they're not constants either. Go ahead and change them if you dare:

```
AF_INET = 30
```

Alas, it's probably foolhardy to expect any modern high-level language to support the full power of C preprocessor macros (e.g., `#define AF_INET 2`). So, people who decide to change constants probably get what they deserve. I digress.

Problems with Constants

Gravitas aside, constants have always presented a number of weird problems for Python programmers. For example, suppose you're using Python 2.7 and you're trying to perform debugging and diagnostics. In your code, the constants are merely presented as their corresponding value. For example, consider this code:

All About That Constant

```
from socket import socket

def create_socket(address_family, socket_type):
    log.info('Creating socket: family=%s, type=%s',
            address_family, socket_type)
    return socket(address_family, socket_type)
```

If you call the function using `create_socket(AF_INET, SOCK_STREAM)`, you'll get a log message that looks like this:

```
INFO:Creating socket: family=2, type=1
```

As you can see, you lose the symbolic names such as `AF_INET`, putting the burden on users to perform some kind of reverse lookup if they want to find out more information. Even doing that is a bit annoying if you don't know what you're doing. For example, do you simply go through the `socket` module constants one-by-one in the interactive interpreter?

```
>>>AF_UNIX
1
>>>AF_INET
2
>>>SOCK_STREAM
1
>>>
```

Or, if you're really stuck, do you pull out some kind of advanced magic to see all of the possible values?

```
>>>import socket
>>> sorted((getattr(socket, name), name) for name in
dir(socket)
...     if name.startswith('AF_'))
...
[(0, 'AF_UNSPEC'), (1, 'AF_UNIX'), (2, 'AF_INET'), (11, 'AF_SNA'),
(12, 'AF_DECnet'), (16, 'AF_APPLETALK'), (17, 'AF_ROUTE'),
(23, 'AF_IPX'),
(30, 'AF_INET6')]
>>>
```

Suppose you wanted to add some kind of enforcement of constant values in your code: for example, making sure the user only provided valid values for the arguments. Maybe you would write something like this:

```
from socket import (socket,
                   AF_UNIX, AF_INET, AF_INET6,
                   SOCK_STREAM, SOCK_DGRAM)

_address_families = { AF_UNIX, AF_INET, AF_INET6 }
_socket_types = { SOCK_STREAM, SOCK_DGRAM }

def create_socket(address_family, socket_type):
    log.info('Creating socket: family=%s, type=%s',
            address_family, socket_type)
    if address_family not in _address_families:
```

```
        raise ValueError('Bad address family %s' % address_family)
    if socket_type not in _socket_types:
        raise ValueError('Bad type %s' % socket_type)
    return socket(address_family, socket_type)
```

Such a solution is kind of verbose and annoying. Moreover, it only “works” until a user comes along and writes the arguments in the wrong order such as `create_socket(SOCK_STREAM, AF_INET)`. Or did they write `create_socket(AF_UNIX, SOCK_DGRAM)`? There's really no way to know. The mind boggles.

Constants in Python 3

Starting in Python 3.4, an interesting thing happened to constants. Fire up a Python 3.4 interpreter and take a look at the `socket` module:

```
>>> # This must be done in Python 3.4
>>>import socket
>>>socket.AF_INET
<AddressFamily.AF_INET: 2>
>>>socket.SOCK_STREAM
<SocketType.SOCK_STREAM: 1>
>>>
```

Notice how the constants now identify themselves by a symbolic name and value. This is very different. Moreover, this change affects everything else. For instance, if you print a constant, you just get the name:

```
>>>print(socket.AF_INET)
AddressFamily.AF_INET
>>>
```

This means that in other code, such as the example involving logging, you'll now get a log message that looks like this:

```
INFO:Creating socket: family=AddressFamily.AF_INET,
type=SocketType.SOCK_STREAM
```

In fact, you can even do a kind of type checking. Consider this slightly modified version of code:

```
from socket import socket, AddressFamily, SocketType

def create_socket(address_family, socket_type):
    log.info('Creating socket: family=%s, type=%s',
            address_family, socket_type)
    if not isinstance(address_family, AddressFamily):
        raise TypeError('Bad address family %s' % address_family)
    if not isinstance(socket_type, SocketType):
        raise TypeError('Bad type %s' % socket_type)
    return socket(address_family, socket_type)
```

In this example, `AddressFamily` and `SocketType` represent all of the valid values for the `address_family` and `socket_type` arguments, respectively. However, this checking is more than just values. It will catch errors such as swapped arguments like this:

```
>>> # Good
>>> s = create_socket(AF_INET, SOCK_STREAM)
>>>
>>> # Bad
>>> s = create_socket(SOCK_STREAM, AF_INET)
Traceback (most recent call last):
...
TypeError: Bad address family SocketType.SOCK_STREAM

>>>
```

Keep in mind, the value of `SOCK_STREAM` is perfectly valid as an address family (it's the same as `AF_UNIX`). Yet, the code caught the error. If you're like me, you'll find all of this to be very interesting.

Enter Enums

Starting in Python 3.4, you can now start defining constants in the form of an “enumeration” class. There are two different flavors, a standard `Enum` and an `IntEnum`. Here are some examples of enum definitions:

```
from enum import Enum
class Color(Enum):
    red = 1
    blue = 2
    green = 3

from enum import IntEnum
class AddressFamily(IntEnum):
    AF_UNIX = 1
    AF_INET = 2
    AF_IPX = 23
    AF_INET6 = 30
```

The first enumeration, `Color`, simply defines a collection of symbolic constants. To refer to them in your code, you just use the class name as a prefix like this:

```
>>> Color.red
<Color.red: 1>
>>> Color.blue
<Color.blue: 2>
>>>
```

Normally, you will just use these names in your code. However, should you need to know the name and value, you can obtain them as attributes as follows:

```
>>> Color.blue.name
'blue'
>>> Color.blue.value
2
>>> AddressFamily.AF_INET.value
2
>>>
```

Such attributes can be useful in situations where you need to convert an enum into a different format or into a value that you might use in an external representation (e.g., JSON). To go the other way, you can use the class name to convert a value back into an enum:

```
>>> Color(2)
<Color.blue: 2>
>>> Color(4)
Traceback (most recent call last):
...
ValueError: 4 is not a valid Color
>>> AddressFamily(2)
<AddressFamily.AF_INET: 2>
>>>
```

As you can see, such conversions are already aware of the valid enum values. If you try to convert a bad value, you'll get an error.

If you want to know all of the possible values of an enumeration, simply turn the class into a list or iterate over it. For example:

```
>>> list(Color)
[<Color.red: 1>, <Color.blue: 2>, <Color.green: 3>]
>>> for val in AddressFamily:
...     print(val)
...
AddressFamily.AF_UNIX
AddressFamily.AF_INET
AddressFamily.AF_IPX
AddressFamily.AF_INET6
>>>
```

In this example, two different kinds of enums were defined. The difference between `Enum` and `IntEnum` concerns their interaction with the rest of the type system and compatibility with the integers.

`Enum` types implement a strict form of type checking that do not allow any kind of mixing with other types or other enums. For example:

```
>>> Color.blue
<Color.blue: 2>
>>> Color.blue == 2 # Notice failed equality
False
>>> Color.blue == AddressFamily.AF_INET
False
>>> Color.blue + 4
Traceback (most recent call last):
...
TypeError: unsupported operand type(s) for +: 'Color' and 'int'
>>>
```

All About That Constant

In fact, the values associated with an Enum type are arbitrary. For example, it would be perfectly fine to define this:

```
class Color(Enum):
    red = 'R'
    blue = 'B'
    green = 'G'
```

Keep in mind that the intended use of the enum would be through the symbolic names such as `Color.red`, not the value. As such, nothing is implied or guaranteed about the capabilities of the enum itself. The value really only becomes useful in code that needs to convert the enum to/from a different type for instance.

The `IntEnum` class, on the other hand, makes an enum compatible with integers. This is especially useful if you're defining constants that need to interface with external libraries or legacy code. Or if the constants need to be used in mathematical operations. For example:

```
>>>AddressFamily.AF_INET == 2
True
>>>AddressFamily.AF_INET + 10
12
>>>
```

`IntEnum` types are also useful if constants are defined in order to perform other operations such as the formation of a bitmask. For example:

```
>>>class Modes(IntEnum):
...     READ = 1
...     WRITE = 2
...     DELETE = 4
...
>>> a = Modes.READ | Modes.WRITE
>>> a
3
>>>
```

Making Enums

Perhaps the most obvious way to define an enum is through a class definition as shown in the example. However, this offers no help to existing code where a large number of constants might already exist. Fortunately, there is an alternate interface involving dictionaries. For example, suppose you have some constants already populating a dict like this:

```
colors = {
    'red' : 1,
    'blue' : 2,
    'green' : 3
}
```

To create an enum, simply call `Enum()` or `IntEnum()` as a function and pass the dictionary like this:

```
from enum import Enum
Color = Enum('Color', colors)
```

If you are clever, you can use this to create enumerations from existing sets of constants. For example, suppose you wanted to make an enum from all of the flags passed to the `os.open()` function. You could simply gather them up using a dictionary comprehension and pass them to `IntEnum()` like this:

```
>>>import os
>>>flagvals = { name:val for name, val in vars(os).items()
...             if name.startswith('O_') }
>>>flagvals
{'O_SYNC': 128, 'O_SHLOCK': 16, 'O_TRUNC': 1024, 'O_CREAT': 512,
'O_EXCL': 2048, 'O_RDWR': 2, 'O_DSYNC': 4194304, 'O_NONBLOCK': 4,
'O_ACCMODE': 3, 'O_WRONLY': 1, 'O_ASYNC': 64, 'O_RDONLY': 0,
'O_APPEND': 8, 'O_NOFOLLOW': 256, 'O_DIRECTORY': 1048576,
'O_NOCTTY': 131072, 'O_NDELAY': 4, 'O_EXLOCK': 32}
>>>Flags = IntEnum('Flags', flagvals)
>>>Flags.O_TRUNC
<Flags.O_TRUNC: 1024>
>>>Flags.O_CREAT
<Flags.O_CREAT: 512>
>>>Flags.O_RDONLY
<Flags.O_RDONLY: 0>
>>>
```

If you were feeling particularly adventurous, you could even patch the original `os` module to use the newly created enums:

```
>>>vars(os).update({f.name:f for f in Flags})
>>>os.O_CREAT
<Flags.O_CREAT: 512>
>>>os.O_RDWR
<Flags.O_RDWR: 2>
>>>
```

Since `IntEnum` classes are compatible with integers, everything should continue to work the same as before except for symbolic names appearing in the event that a flag value is ever printed or logged.

The Normal Rules Don't Apply

Having introduced enums, most Python programmers will find them to behave in all sorts of ways that are quite different from normal class definitions. For example, duplicate entries result in an error:

```
class Color(Enum):
    red = 1
    blue = 2
    red = 3      # An error. Duplicate.
```

Enum classes always keep their entries in the same order as listed:

```
class Color(Enum):
    red = 10
    blue = 9
    green = 8

cols = list(Color) # [ Color.red, Color.blue, Color.green]
```

You can't inherit from an enum:

```
class MyColor(Color):
    purple = 4      # Error. Can't extend Color
```

And the members of an enum can't be redefined:

```
Color.red = 4      # Error. Can't reassign members
```

The members of an enum are also instances of the class itself:

```
>>> isinstance(Color.blue, Color)
True
>>>
```

All of this unusual behavior is the result of enums being defined through advanced features of Python metaclasses. It's not possible (or really necessary) to dive into the details here, but if you've ever wondered about the power of Python metaprogramming, enums are a good example of what's possible.

Final Words

As a new Python feature, enums are not something you're likely to encounter in much code. However, they are starting to be used in various places in the standard library and will likely have increased usage in future Python versions. In my own application code, I often find myself defining various sorts of constants to indicate modes, flags, and similar kinds of functionality. With the addition of enums, I'm now starting to think that they might be a useful way to provide improved debugging, type safety, and other similar features. Although enums first appeared in Python 3.4, the `fluf.enum` package can be used to add them to earlier versions of Python including Python 2.7.

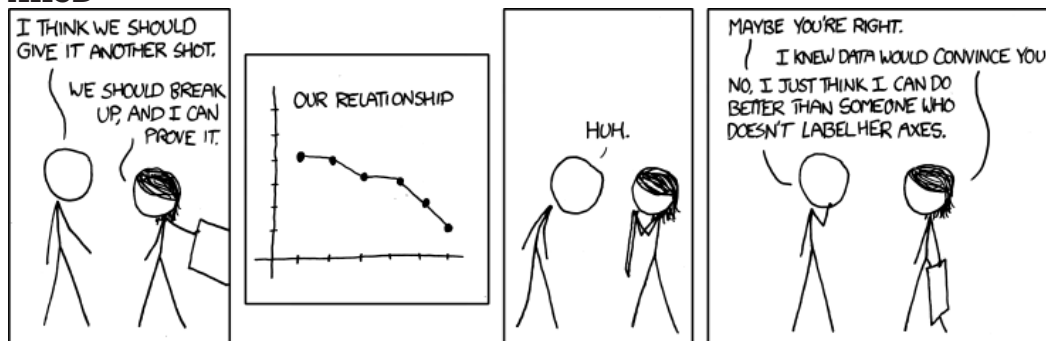
Resources

<https://docs.python.org/3/library/enum.html> (official documentation for enums).

<http://legacy.python.org/dev/peps/pep-0435/> (adding an Enum type to the Python Standard Library).

<https://pypi.python.org/pypi/fluf.enum> (an enum implementation compatible with Python 2.7).

XKCD



xkcd.com

iVoyeur Rediscovering collectd

DAVE JOSEPHSEN



Dave Josephsen is the sometime book-authoring developer evangelist at Librato.com. His continuing

mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

I recently saw Shadaj Laddad’s talk at this year’s OSCON, entitled “The Wonders of Programming” [1]. If you haven’t had the pleasure, Shadaj is a 14-year-old programmer who (among many other things) wrote a bioinformatics Scala library. In the talk, he describes how he was encouraged to program computers from the age of six, and he gives helpful tips to parents and other children who are interested in pursuing computer programming.

It probably doesn’t surprise any of us that Shadaj credits Lego Mindstorms as having an early impact on his understanding of systems programming. Mindstorms are, of course, actually programmable (in myriad languages), but I think many engineers come back to Legos in general when asked about toys that awakened in them a love of science, technology, engineering, mathematics, and, more generally, building things and solving problems.

In my own childhood, if Legos ever became boring, they didn’t remain so for long. Again and again, as my interests changed, Legos always found a way to become relevant again. I would rediscover them when I needed a ramp to jump hot-wheels, or when we were one blaster short, and wanted to reenact *Star Wars Episode 4* from memory (an almost daily occurrence among my third-grade friends). Later, I would rediscover them when I needed just the right-sized wedge to keep my Commodore tape drive functioning or a box to house an 8088 project to prevent it from grounding out. Even last week, I rediscovered them when I was looking for a clever way to keep track of the myriad groupings of household keys [2].

Maybe it’s silly, but I’ve often wondered over the years of using and implementing little UNIX tools that do one thing well, or more recently, Web-based micro-services architecture, what percentage of systems engineering tools and practices we owe to Legos. To be sure, modular, single-purpose primitives that can be combined to form more complex entities is just one of many design methodologies, and it’s an obvious one that no doubt predates the actual creation of Lego by several thousand years.

How many times have we reinvented the monitoring system? How many times have we redefined what a monitoring system even is? I couldn’t tell you, being not disposed to anthropology myself. I can tell you, however, that every time someone finds a problem for which the commonly adopted monitoring systems aren’t well adapted, that person usually winds up implementing a set of primitives that meets the need. When we build new monitoring infrastructure, it seems we inevitably rediscover the building blocks, and whenever this happens, silly or not, I have to admit it feels exactly the same as rediscovering my Legos.

I’ve been working a lot with collectd lately, a project that, were monitoring systems Legos, would probably be a valued and coveted block. At my day job, we’ve just finished implementing some service-side, turnkey support for it, to remove dependencies and make it easy for our customers who happen to use collectd to ship their measurements to us out of the box, so I’ve been playing with collectd a great deal over the past few weeks.

Having not used it for a few years, I'd forgotten what a nifty tool it is, and having rediscovered this particular Lego block has been a lot of fun for me. Looking through my GitHub repo of articles, I'm surprised to find that I've never actually written about collectd here, which is an oversight I'd like to correct now.

(Re)Introducing Collectd

Collectd is a modular metrics collection daemon written in C. Collectd loops through a list of user-specified plugins, executing each to gather performance metrics from the OS, or locally running user-space processes. Once gathered, collectd outputs these metrics on a set interval using one or more output plugins to targets like log files, aggregation daemons like StatsD, and metrics-processing systems like Librato. Collectd is a great way to begin collecting data; it offers a ton of useful metrics for a very small operational investment.

Collectd is a great fit for you if:

- ◆ You want a flexible standalone collection agent to collect performance metrics from your systems using the standalone agent pattern.
- ◆ You're running Virtual Machine instances and want to grab per-instance CPU/Disk/Memory metrics.
- ◆ You want a simple way to collect metrics from running server processes like MySQL, Apache, Redis, Nginx, or MongoDB.
- ◆ You're looking for a well-documented, widely used and trusted open-source collection agent that is available on most Linux distributions.

Installing Collectd

Collectd gets installed on every system you want to monitor, and it's pretty simple to install. It runs as a standalone daemon process and is configured by way of a classical UNIX conf file in `/etc/collectd`. You can obtain and build collectd from source, but packages exist for all major distros, and most small ones. For example, on a Debian-based system, you'd enter

```
apt-get install collectd
```

How Does It Work?

Collectd generally follows the standalone agent pattern. It runs on every host you want to monitor, and it either reports directly to an upstream metrics aggregator or writes metrics to the local file system.

Starting the Daemon

Debian-based distros start collectd automatically when you install it, but if collectd isn't already running, you should be able to start the daemon using the appropriate init method for your OS, or directly by executing `collectdmon`. If collectd won't run, or

The Agent-Based Pattern

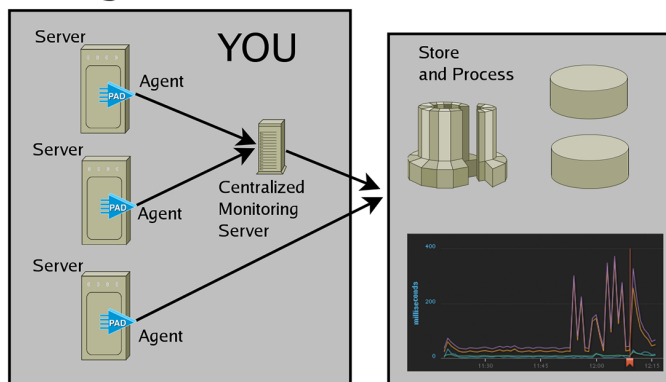


Figure 1: Collectd works as a standalone collection agent, indicated by the PAD icons in this figure.

if it appears to be constantly restarting, you can run it manually with an `-f` switch, which will prevent it from forking into the background.

Collectd also includes the `-t` switch, which tests the validity of the configuration file and is helpful for troubleshooting startup problems.

Plugins

Collectd's behavior is dictated by two types of plugins. Input plugins gather performance data from the OS or applications running on the system. The CPU input plugin, for example, interacts with the OS to measure the same CPU-related metrics returned by the UNIX `top` command, like the percentage of time the CPU spends executing user-space processes or waiting on I/O.

The Nginx plugin, by comparison, queries a running Nginx server to gather metrics like the current number of requests and connection information. Users are encouraged to write their own plugins to pull data from specific resources and contribute them back to the project so others can benefit from them.

Output plugins are then used to send the gathered metrics data to other services for storage or analysis. The `write_http` plugin is one example of an output plugin, sending metrics data to a remote Web server in the prescribed JSON format. Other output plugins support graphing systems like RRDtool, the AMQP message transport, or even humble CSV files.

Many plugins exist for collectd. The default collectd installation on the current Ubuntu LTS (Trusty) comes preconfigured with 100 plugins, 14 of which are automatically enabled. To give you a feel for the sorts of metrics that are collected out of the box, here are all of the input plugins that were enabled on my test Trusty box by default:

iVoyeur: Rediscovering collectd

- ◆ battery: for systems with internal batteries like laptops
- ◆ cpu: CPU stats (%wait, %user, etc.)
- ◆ df: file-system capacity (e.g., inodes free)
- ◆ disk: disk performance (I/O per second)
- ◆ entropy: measures the effectiveness of the PRNG
- ◆ interface: network interface (I/O per second)
- ◆ irq: times per second the OS has handled an interrupt
- ◆ load: 1, 5, and 15-minute load average
- ◆ memory: RAM usage
- ◆ processes: number of processes grouped by state (running, sleeping, stopped, etc.)
- ◆ swap: swap capacity and usage
- ◆ users: number of users currently logged in

Plugin Configuration and Dependencies

For each plugin that collectd loads, there is a LoadPlugin line in the collectd.conf file. Some plugins require only this line, although most require some additional configuration to do things like specify formats or locate files or directories in the file system.

A few plugins depend on other plugins to operate. A notable example is the JMX plugin, which requires the Java plugin to function. Settings and dependency information for each plugin are fully documented at the collectd wiki.

Mind the Polling Interval

Collectd's polling interval is controlled by the Interval attribute in the collectd.conf file. Because many upstream visualization tools make assumptions based on this interval, you should think carefully about your desired resolution, set it once and avoid changing it, and take steps to ensure that this setting remains the same on every host.

Modifying collectd's polling interval will affect the resolution of your metrics in upstream visualization systems. Some systems handle this better than others. RRDtool, for example, is heavily dependent on a preconfigured polling interval, so changing this setting could render your existing RRDs inoperable. Again, set it carefully, and then leave it alone.

Rollups with Collectd's Network Plugin

With collectd's network plugin, it's possible to specify one or more collection servers, to which all hosts emit their metrics. This can simplify per-host configuration and minimize network access control permissions, providing a means to aggregate and proxy a site-wide metrics stream by configuring the server to write to an upstream service like Librato.

A friend once told me that usually engineering was about building things, but that sometimes it was also about destroying things, to see what can be made from the parts. I think I might add that sometimes our job is to play with different kinds of parts, because playing with parts teaches us about how things could be built.

Being a building block, collectd isn't going to replace a monolithic monitoring system like Sensu or Nagios, and I'm certainly not advocating that you destroy a functional monitoring system only to rebuild it on collectd, but because most (if not all) monitoring systems can be made to accept data from collectd, it's worth playing with, whether you're already running a monolithic system or just trying to figure out what pieces fit with what. Even if you've played with it before, you might learn something new. I did.

Take it easy.

References

- [1] "The Wonders of Programming": <http://www.oscon.com/oscon2014/public/schedule/detail/35956>.
- [2] A Lego keyholder: <https://www.youtube.com/watch?v=5TdYhkVutkQ>.



Do you have a USENIX Representative on your university or college campus?

If not, USENIX is interested in having one!

The USENIX Campus Rep Program is a network of representatives at campuses around the world who provide Association information to students, and encourage student involvement in USENIX. This is a volunteer program, for which USENIX is always looking for academics to participate. The program is designed for faculty who directly interact with students. We fund one representative from a campus at a time. In return for service as a campus representative, we offer a complimentary membership and other benefits.

A campus rep's responsibilities include:

- Maintaining a library (online and in print) of USENIX publications at your university for student use
- Distributing calls for papers and upcoming event brochures, and re-distributing informational emails from USENIX
- Encouraging students to apply for travel grants to conferences
- Providing students who wish to join USENIX with information and applications
- Helping students to submit research papers to relevant USENIX conferences
- Providing USENIX with feedback and suggestions on how the organization can better serve students

In return for being our "eyes and ears" on campus, the Campus Representative receives access to the members-only areas of the USENIX Web site, free conference registration once a year (after one full year of service as a Campus Representative), and electronic conference proceedings for downloading onto your campus server so that all students, staff, and faculty have access.

To qualify as a campus representative, you must:

- Be full-time faculty or staff at a four year accredited university
- Have been a dues-paying member of USENIX for at least one full year in the past

For more information about our Student Programs, contact
Julie Miller, Marketing Communications Manager, julie@usenix.org

www.usenix.org/students

For Good Measure Stress Analysis

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

Reality is the leading cause of stress amongst those in touch with it.

—Jane Wagner

In retrospect, the financial collapse of 2008 had useful side effects. We in distributed computing can be rightly thankful that it was the financial services world that shouldered the task of proving that (we) humans are in fact entirely capable of building systems that (we) humans cannot then understand well enough to stably operate. I say thankful as (1) it wasn't us, and (2) money lost can be replaced with money printed, but not everything is so fungible.

One of the useful side effects was that of ratcheting up the compulsory simulations of bad events. Various sovereigns require these; most are called "stress tests." What these stress tests propose to do is to show how the largest bank holding companies (BHCs) would fare in the event of various unhappy financial events in general, things that are "shocks to the system" for which the BHC must either be able to absorb or be invulnerable to the contagion.

I've long considered financial services to be the avatars in cybersecurity simply because the financial world differs from every other industrial sector in that the bigger the bank, the greater the percentage of its business is done with competitors (i.e., BHCs are mutually dependent). I'm here to suggest that it is that mutual dependence that generates risk of the sort that stress tests exist to measure.

In another column long ago, I tried to explore the idea of a "margin of safety" for cybersecurity, something on par with how a civil engineer thinks about bridge failure under load. Cryptography has long had such concepts. I now think that the stress test route is the one for cybersecurity to follow. In a way, some already do this—including contingency plans for data breaches that involve reverting to paper while evidence is gathered [1].

We all know that organized crime and military powers alike have both tools for mass disruption and tools for precision targeting. With that said, the pervasive interdependence of the current Internet sphere is certainly on par with the interdependence of financial markets. The time constants (speed) of the exchange of data and control between major cyberinfrastructures are smaller (faster) than everything else on the planet excepting, perhaps, financial services engaged in high-frequency trading.

So the obvious question is what sorts of simulations might be appropriate metrics for assessing public risk to private yet critical components of the Internet ecosystem? And to whom might a requirement for cybersecurity stress testing apply? As to the latter, in finance the stress tests are required of "systemically important financial institutions" (SIFIs), which include not only banks but also insurance companies and market infrastructure providers [2].

As to the question of what simulations and cybersecurity metrics might be appropriate, in finance the scenario for 2012 stress testing [3] was

- ◆ a peak unemployment rate of 13 percent,
- ◆ a 50 percent drop in equity prices, and
- ◆ a 21 percent decline in housing prices.

If applying such scenarios to major financial institutions represents our best available analogy for what to do in cybersecurity, then how can we in cybersecurity proceed?

The cybersecurity equivalent of the SIFI would include the most important transport providers (ISPs), cybersecurity product suppliers, identity providers, intelligence acquirers and analysts, and any of the XYZ-as-a-service suppliers as have clients who are themselves critical infrastructure players and for which security is part of the claimed package of service benefits. It is likely that in cybersecurity we'd have a longer list than the eight domestic and 32 global SIFIs, because for finance it is easy to tell who to include by the "too big to fail" test, whereas for cybersecurity the web of dependencies is more like looking for those entities that are "too interconnected to fail."

The cybersecurity equivalent of the stress test would not be just one scenario but, rather, a number of scenarios. Let me suggest, however, one sample parallel to that which is applied to the SIFIs, viz., the simultaneous appearance of

- ◆ a vulnerability requiring client-side reinstallation for 25 percent of all endpoints,
- ◆ a sustained 50 percent drop in available bandwidth, and
- ◆ the wholesale data loss of a top-three cloud provider.

As with the banks, the question is whether the enterprise being stress tested can survive in the above scenario. Stress tests are fundamentally different from the tests (and associated metrics) that come from such disparate things as static analysis of code, penetration testing, cryptographic strength assessment, and so forth. In every case with which I am familiar, the tests we currently do are designed to answer the question, "Am I or my clients at risk from things that I am supposed to directly control?" The tests I am proposing answer a different question: "Can I withstand the failure of others on whom I depend?" That gets to the very heart of risk—a dependence on the expectation of system state.

I would like to work with a number of interested parties to come up with a set of scenarios, a set motivated by the systemic risk to those other entities that depend on the cybersecurity industry and which would ask questions in the same spirit that the stress tests mandated by Basel III [4], Dodd-Frank [5], the European Banking Authority [6], and so forth, ask: Can the institutions be made to survive cyber-failure scenarios through the application of cybersecurity techniques that we already have in hand, or not?

The stress testing of financial institutions could not have come into force without the near approach of general collapse on a global scale. Must we hope for the near approach of general collapse on a global scale within the cybersecurity infrastructure? One wishes otherwise, but if such a crisis does occur, then we cannot let it go to waste. Thinking and writing about what a useful cybersecurity stress test regime would contain is the best way to avoid letting the coming crisis go to waste. With the possible exception of finance, no part of modern life offers the chance of common mode failure as much as cybersecurity does [7]. It is our duty to realistically measure that risk and to prepare or preserve alternate paths. Finance has, unwillingly or no, blazed a trail. What will we do?

References

- [1] US restaurant chain P.F. Chang's China Bistro reverted to manual credit card imprinting during investigation of a security breach that allowed hackers to steal customer payment card data from multiple stores: arstechnica.com/security/2014/06/pf-chang-turns-to-vintage-1970s-tech-after-credit-card-breach/.
- [2] "List of systemically important banks," Wikipedia: en.wikipedia.org/wiki/List_of_systemically_important_banks.
- [3] Federal Reserve Board of Governors, Comprehensive Capital Analysis and Review: www.federalreserve.gov/newsevents/press/bcreg/20120313a.htm.
- [4] Bank for International Settlements, International Regulatory Framework for Banks(Basel III): www.bis.org/bcbs/basel3.htm.
- [5] US Senate Committee on Banking, Housing, & Urban Affairs, Brief Summary of Dodd-Frank Wall Street Reform and Consumer Protection Act: www.banking.senate.gov/public/_files/070110_Dodd_Frank_Wall_Street_Reform_comprehensive_summary_Final.pdf.
- [6] European Banking Authority, EU-wide stress testing: www.eba.europa.eu/risk-analysis-and-data/eu-wide-stress-testing.
- [7] Dan Geer, "Heartbleed as Metaphor," *Lawfare*, April 21, 2014: www.lawfareblog.com/2014/04/heartbleed-as-metaphor.

/dev/random Buying Snake Oil

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

While I only stood in line for about 45 minutes in order to accomplish it—as opposed to camping for a day or two in a butt-grabber nylon chair on the sidewalk—I got my iPhone 6 today. (Today being September 22, no matter what *your* calendar says; we’ve talked about publishing relativity before.) I only got the 16 GB version and not the Plus model, either, but announcing this cherry acquisition is not my *raison d’être* here.

As I was biding my time in the store I wandered over and started playing with a Surface Pro 3. Completely contrary to my expectations, I really liked the feel and functionality of the thing. I may just have to get one if my novels ever start to sell the way my publisher assures me will happen (seconds before the mile-wide asteroid that got by NASA impacts). Of course, if the Surface ran, say, Ubuntu it would be even better (yes, I’ve seen the *Geek* article on accomplishing this). I fondled it for a while and then in order to express my profound amazement at having enjoyed the experience bought *Minecraft* for the Xbox 360. I’m sure the logic in this is obvious to you all.

Moving on, I recently stumbled across this parody ad I wrote some years ago: I suspect merely to drain a modicum of anger from my psyche, as I do that from time to time. It’s rather dated—you can tell by the “geek code” reference—but it nevertheless conveys a sentiment I still believe to be valid. If there *is* a “Carpe Diem (In)Security Systems,” out there now, by the way, I can’t imagine *what* you were thinking. It isn’t about you, anyway, so don’t get your knickers in a knot.

If you’re offended by something in this little diatribe, get over it now, before you even start reading. Satire is *supposed* to be offensive, or at least aggressively thought-provoking: That’s how it gets its point across. There was a time when I produced pretty much nothing but satire and parody. One of my friends, an accomplished poet, even went so far as to call me a “Master of Parody.” I probably wouldn’t go that far unless you encouraged me (twist my arm: please), but I do remember wanting very much to write for the *Harvard Lampoon* as a teen, until I found out you had to go to, well, *Harvard* to do that. I could barely afford the local junior college, much less an Ivy League school.

This fictitious ad reveals my own mental state rather than shining any critical light on the security industry at the time, admittedly, but there are some telling depths to be plumbed. For example, the Philip K. Dick reference to arresting people for what they were thinking ensures that this was written after 9/11 because prior to that I considered the “PreCrime” concept to be pure, rather absurd, science fiction. Once we started passing classified laws that average citizens weren’t even allowed to know about, much less defend themselves against when charged, I reassessed that evaluation and came to the conclusion that PKD was, rather than simply mad, disturbingly prescient as well.

Does the Psychic Network even *exist* anymore? They should have seen that coming...

CD(I)SS Announces PVA: The Future of Digital Defense

It is with considerable pride, anticipation of media coverage, and downright old-fashioned avaricious glee that we here at Carpe Diem (In) Security Systems announce a major breakthrough in vulnerability reporting. For years the software industry has relied upon the hopelessly outdated reactive paradigm, in which independent investigators spend many long hours combing through source code looking for unbounded variables or hitting new releases of software with everything including the kitchen sink in an attempt to lay bare a potential weakness and thus make a name for themselves in the voracious canine-consume-canine world of People Who Call Themselves Infosec Experts. But now, thanks to the amazing skills of our 7337 h4x0r squad at CD(I)SS, we have taken that tired old paradigm to new, never-before-imagined heights.

After years of tedious and meticulous labor, the men, women, et Al (we had to let Al tag along, as his mom won't be home until 5:30) of CD(I)SS have developed the Preemptive Vulnerability Announcement (PVA). Taking our cue from time-honored traditions such as arresting people because they *might* be *thinking* about committing a crime, our team of slavishly devoted security analysts with no discernible personal lives have devised a method for evaluating the security posture of software *before it is even written*.

Yes, by boosting the latent psychic abilities of our analysts with a secret blend of eleven herbs and spices (with emphasis on the herbs), CD(I)SS is able to foresee and release vulnerability reports *before the software's designers*

commit the first line of code. Not only will we spell out all of the security flaws in the future product, subscribers to our Platinum Iridium Gallium Arsenide Service will receive a patch that can be applied to the code when it is finally released, no matter what language is used. The odds are it won't provide any useful function, but you'll probably never know that. Why? Because, as it proudly says in our Mission Statement, *People are Stupid*. (Fond nod to Terry Goodkind.)

How can we offer such incredible service to our customers? It's simple, really. We know from expensive but questionable market surveys that many of you patronize the Psychic Network, make theologically and ethically shaky televangelists nevertheless wealthy, and believe those commercials that insist you must call within the next 10 minutes or you won't receive, absolutely free, a second 12-ounce bottle of Oxy-Stain (with free disposable container): Not Sold in Any Store. With that kind of consumer gullibility already in place, making the jump to PVAs was child's play. It was certainly the next logical development—but it remained for our dedicated group of 7377 h4x0rs to take that profitable step. Did we mention that we hire only pure-bred, soda-swilling, certified 7377 h4x0rs? Your geek code must be at least 256 characters long even to get an interview.

Best of all, no one will ever again scoop us (or our investors) on a vulnerability announcement. If that doesn't give you a warm fuzzy, there's something wrong with you. But then, if you're willing to accept our claims at face value, that pretty much goes without saying.

CD(I)SS: Where your credulity is money in our bank.

Book Reviews

MARK LAMOURINE AND RIK FARROW

Functional Thinking

Neal Ford

O'Reilly Media Inc., 2014; 179 pages

ISBN 978-144936551-6

Reviewed by Mark Lamourine

Functional programming has, for a long time, been the realm of the theorists, the purists, and the AI specialists. Derived directly from the lambda calculus, the mathematical underpinnings of computation theory, functional programming has always felt like it required a different mental model. FP was an alien world. It didn't seem like the concepts could be applied without throwing out everything I know and adopting a new programming language.

While it's true that some languages make pure functional programming easier than others, many languages today provide the most fundamental feature of functional programming: functions as first class objects. This means that it's possible to apply the concepts of FP even in languages that are not pure functional languages. Ford does use languages, Scala, Groovy, and Clojure, that illustrate his ideas clearly, but he also demonstrates code in Java 7 and 8.

Ford really wants the reader to begin to look at certain classes of coding problems differently. Each chapter has a word or phrase that is used to guide the examination. I'm not sure how effective they are really but they may work for some.

In the first chapter, "Shift," Ford talks about recognizing list processing opportunities. He introduces the MapReduce pattern and filtering using chained list processing functions. Ford shows in following chapters how to use higher order functions and recursion to replace iteration ("Cede"), and how to use memoization to create "lazy" data structures to delay processing ("Smarter, not Harder"). In "Evolve" Ford introduces Clojure and the concept of replacing defined data structures with dynamic functions that both manipulate and represent the program state. The final two chapters, "Advance" and "Polyglot and Polyparadigm," introduce design patterns suited to functional programming and examples of mixed-style languages and programming practices.

I admit I'm not a convert to pure functional programming. In the examples given, the code is indeed often more concise than the comparable object oriented or imperative style. To my aging eye, the results often smack of cleverness, which can obscure the intent of the author. I've seen and written long strings of chained functions on lists, and they often seem to reach a point where the meaning is no longer evident to the reader.

I'm also not a fan of the convention of creating a new function for every possible variation of operation on some data structure. The results are an ever increasing catalog of minutely specialized functions that would otherwise be a clean set of methods on a class.

Given all that, I do have a newer appreciation of functional programming, and I'll keep an eye out for opportunities to apply what I've learned. *Functional Thinking* has given me a perspective on functional programming techniques and philosophy that I missed when I learned my first functional languages (Common Lisp and Scheme) in college. I do wish I'd had some of this perspective then.

SDN: Software Defined Networks

Thomas D. Nadeau and Ken Gray

O'Reilly Media Inc., 2013; 353 pages

ISBN 978-1-449-34230-2

Reviewed by Mark Lamourine

I pick up most books from O'Reilly today expecting to breeze through the introduction and the first few chapters at least. The authors of *SDN* made me work. The subtitle of the book is "An Authoritative Review of Network Programmability Technologies," and I think they live up to it.

Nadeau and Gray have a difficult task, too. To discuss the technologies that can be used to create and manage programmable networks, the reader needs first to have some understanding of the network components themselves. While many sysadmins are familiar with the use of Layer 2 (switching) and Layer 3 (IP, routing) devices and the data line protocols, fewer are familiar with the internal architecture and components of these devices. To make things more difficult, most of those components have proprietary names and acronyms or abbreviations.

The authors move very quickly through this first section and don't make many concessions to the networking novice. They introduce the concept of a "Data Plane," in which the network payload moves, and a "Control Plane," which defines the characteristics of the network. The key concept that drives the rest of the book is the idea of a distributed control plane. In hardware-defined networks, the control plane is restricted to the individual switch or router device. In a few kinds of device, blade or stacked switches and routers, the control plane is abstracted one level away, but never farther. The abstraction that software offers opens up the possibilities. In the extreme case, the control plane could be completely centralized. The topology variations and their effects on the construction of a network, with the benefits and flaws, fill the remainder of the chapter.

This first section lays the necessary groundwork, but I would have appreciated a bit more help with understanding the typical components, their relationships and interactions in a static hardware network. This is the base on which the rest of the book is built. I think they could have spent some more effort to make this truly complex topic clearer.

The second chapter introduces OpenFlow, which the authors use as a touchstone. It sets a baseline against which the rest of the software in the book is compared. OpenFlow is the result of the first attempt at a software-defined network. While it has gained support both from corporate and open source contributors, it is largely acknowledged to be insufficient to create a complete network by itself.

Nadeau and Gray proceed to move from the lowest level of network control up to building complete virtual network topologies on top of the same physical hardware that carries packets from one place to another. This is a technical review, so the authors list and describe the software capabilities and characteristics, not how to configure or use them in operations.

Software-defined networks are an incomplete and evolving subject. The body of the book alternates between lists of vendor and open source technologies and some discussion of how they fit into a programmed network. The end of each chapter is a short section in which the authors sum up what we've learned and where the gaps still are.

This book does a good job of illuminating the state of software-defined networks once the authors get past the details of the characteristics of distributed or centralized control. Unfortunately, the field has changed dramatically even in the year since it was published. The introduction (and withdrawal) of OpenStack Quantum and the stumbles of OpenStack Neutron are just a couple of the developments in the SDN space. There are signs sprinkled throughout the text that the authors recognize this and plan to issue updates, but it's not clear when.

Given the current movement toward cloud computing, I've concluded that the role of a sysadmin is expanding rather than contracting. The operator of an OpenStack or commercial cloud service will need to have at least some expertise in all of the traditional enterprise IT silos: Compute, Storage, and Networking. A book like *SDN* might be the best way for someone who needs to get a handle on the problems and possibilities to get conversant.

The Practice of Cloud System Administration

Thomas Limoncelli, Strata R. Chalup, Christina J. Hogan
Addison-Wesley, 2015, 524 pages
ISBN: 978-0-321-94318-7

Reviewed by Mark Lamourine

System administrators are not known for consensus and conformity. It doesn't take long for new admins to fall in love with a tool or a programming language (or to fall into hate). The Editor Wars are probably the most well known ongoing dividing line, but faults can appear around any choice we can make.

This is what makes the books by Limoncelli, Chalup, and Hogan (LC&H) so remarkable. If you ask most sysadmins what single book they should read, the answer will almost certainly be *The Practice of System and Network Administration*. They're going to have a harder time now, with the release of volume 2: *The Practice of Cloud System Administration*. (Just so you know, it's already known by the abbreviation TPOCSA.) I think this is likely to become a must-read.

One of the tenets of TPOCSA (and of all quality design) is "Keep it Simple." The authors present cloud administration in two parts. Pretty simple, eh? First, they define the characteristics of their ideal system, then they go on to describe the methods that they use to try to achieve that ideal.

When I say "describe the system" I mean that in a somewhat abstract way. LC&H aren't talking about which database is best or how much memory you need to render a movie frame. It turns out that all large-scale distributed systems have a set of common characteristics. These, along with the requirements for high reliability and robustness, have led to a set of best practices that have become generally accepted, largely because they have been shown to work. The hitch is that most of them seem counterintuitive and nearly all directly contradict standard practices of two decades ago.

In this section the authors also make clear the scope of what "system administration" means. Up until the advent of virtualization and ubiquitous high-speed networks, it meant OS installation and some network configuration. When the machine was ready it would be handed off to some application and operations team for the rest of the lifetime of a host. The SA tasks would probably include backups and periodic patching (or at least that's what many people thought). Today system administration and operations are largely synonymous. This union even has a word: DevOps (which is contentious, so I won't discuss it further here).

So we're talking about a large-scale distributed system. Whenever you have something big and made up of lots of parts, you inevitably have failures. Much of the rest of the book consists of ways to make that not matter, taking human nature and the "physics" of highly complex systems into account to make robust seamless services that run well even as they are changing.

Scanning over the chapter headings after the section break, I am struck by something that should have been obvious. This is a book about practices. The first section is really a glossary, a base of terminology and concepts on which to build. But what we build from them, the system that results, isn't just the cloud application. The infrastructure that LC&H are talking about here is as much a social one as it is technical. Each of the computational components is meant either to facilitate human communication or to remove painful, time-consuming, or error prone tasks.

System administrators are no longer just brick layers and janitors. They are involved in every phase of application life cycle from inception to a long continuously evolving life span. LC&H discuss the philosophy and practice of each phase, always considering that humans are expensive (and error prone) while computation is cheap. Automation, documentation, and monitoring are all reconsidered with an eye to minimizing drudgery and false rigor and replacing it with a min-set that will evaluate what's really important: comprehension and communication.

I read Gene Kim's *The Phoenix Project* not long after it came out, and while I smiled and nodded knowingly all the way through, it felt a little like a unicorn story. I thought, "This is nice, but no one in business is going to take a novel seriously as a model for business practice." Of course I was wrong, but I still think that something more is needed, not just a parable but a manual. The line where the authors cite Gene for "inspiration and encouragement" indicates that LC&H thought so, too.

There really wasn't much in this book that was new to me. I think much of what's here is already fairly common knowledge. What TPOCSA has done is to bring together in one place the accumulated body of knowledge that has been growing and changing since the birth of the Internet. Today's computer systems are a far cry from the mainframes, minicomputers, and PCs that dominated the 1990s. There have been a number of movements triggered by the changes since then: Agile development, the DevOps movement, Continuous Integration and Deployment. TPOCSA brings them all together and reminds us that the methodology, the philosophy, the ideology are not what matters. The system, running and serving reliably, is what matters. All of the rest are just means to that end.

So who should read it? I think anyone claiming to be a system administrator today should be conversant with what's here, but I think the bigger impact will come when we pass it to a colleague, whether a developer or a manager. There's a lot of confusion around what cloud computing means, and TPOCSA gives us a common base on which to build our systems and our processes.

What If? Serious Scientific Answers to Absurd Hypothetical Questions

Randall Munroe

Houghton, Mifflin, Harcourt, 2014; 305 pages

ISBN 978-0-544-27299-6

Reviewed by Rik Farrow

You are likely familiar with *xkcd*, the comic strip that uses stick characters to great effect. You may not have heard of another project by Munroe, where he answers questions submitted to him, using math and research, to provide sound answers to some very strange queries. I had encountered a couple of Munroe's posts while searching *xkcd.org*, looking for potential cartoons to decorate a *login:* issue, and learned that he was publishing an entire book of answers.

Well, not quite. Munroe interspaces his answers with sets of questions that he wouldn't answer, adding more humor to his book. And like the *Mythbusters*, Munroe will often take a question, provide an answer, then scale the question up, to prove a point, like how many people with laser pointers would it take to light up the dark portion of the moon, or his hair dryer that has ten settings, each using an order of magnitude more power. Munroe uses scaling and statistics in ways that are effective.

I've used Munroe's *What If?* as a great way of taking a break away from my computer, and think it would be an appropriate gift to most geeks and/or scientists that you may have in your life. You might also be able to use Munroe's writing as inspiration for how to answer those ridiculous questions you may have been asked by your management, although I do advise caution in these circumstances.

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

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

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

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

Discounts on registration fees for all USENIX conferences.

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

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

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

USENIX Board of Directors

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

PRESIDENT

Brian Noble, *University of Michigan*
noble@usenix.org

VICE PRESIDENT

John Arrasjid, *VMware*
johna@usenix.org

SECRETARY

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

TREASURER

Kurt Opsahl, *Electronic Frontier Foundation*

DIRECTORS

Cat Allman, *Google*

David N. Blank-Edelman, *Northeastern University*

Daniel V. Klein, *Google*

Hakim Weatherspoon, *Cornell University*

EXECUTIVE DIRECTOR

Casey Henderson
casey@usenix.org

Team USA Brings Home the Gold from IOI 2014

Brian C. Dean, Associate Professor, School of Computing, Clemson University, Director, USA Computing Olympiad

Watching the live scoreboard for a five-hour competitive programming contest can be, as one might imagine, somewhat dull. Unless Scott Wu is part of the contest, that is. This past summer, Scott and three other USA team members competed at the 26th annual International Olympiad in Informatics (IOI) in Taipei, Taiwan—the world's most prestigious computing contest at the high-school level. Although the contest takes five hours on each of two separate days, Scott finished day one with a perfect score in only two hours, and day two with a perfect score in only three hours—an absolutely remarkable performance beyond what anyone at the event, including the judges, had anticipated. Aside from Scott's perfect score and first-place finish, his three other teammates, Steven Hao, Andrew He, and Joshua Brakensiek, all earned gold medals, placing in the top 7% of more than 300 competitors from roughly 80 countries; China was the only other country receiving four gold medals. It was perhaps the best showing ever for team USA over more than two decades of participation in the IOI.

The road to the IOI and programming gold started many years ago for the members of team USA 2014, with participation in the USA Computing Olympiad (USACO). The USACO, a grateful recipient of USENIX sponsorship, provides online programming competitions and algorithmic training materials in which thousands of high-school students take part. Beginning students who are just learning to program start out in our bronze division, where problems require minimal algorithmic training beyond the ability to sort. Top performers in the bronze

division are promoted to the silver division, with problems that help students learn about standard algorithms and data structures (a sample problem is included at the end of this article for those interested). With sufficient effort, students then graduate to the gold division, with problems that would challenge even graduate-level students of computer science, requiring sophisticated algorithmic techniques, clever insight, and plenty of coding experience to implement properly. Based on the result of six monthly programming contests held throughout the academic year, the USACO identifies the top two dozen high-school computing students in the USA and invites them to a rigorous summer “training camp” at Clemson University for additional instruction. The top four from this camp are selected to represent the USA at the IOI.

Of the USA team members competing this year at the IOI, Scott had attended the USACO summer training camp for the past four years, Steven and Joshua for the past three years, and Andrew for the past two years. The camp experience is designed to benefit first-time attendees as well as veterans such as these students, with first-timers receiving a more lecture-centric curriculum, and returning students experiencing a curriculum based more on practice competitions. Beyond the core instructional activities at camp, finalists take part in recreational activities, excursions, enrichment lectures, and innovative computational “game” challenges. For example, this summer students worked in teams to write programs that would bid against each other in a simulated prediction market that would reward the programs best able to predict the winner of a game of chess unfolding in real time. Camp instructors are mostly USACO alums who are now undergraduate and graduate computer science students at top universities, who provide mentorship not only in computational problem solving, but also in the exciting opportunities for advanced study in a variety of computational disciplines. In short, the USACO helps to identify, train, and inspire our next generation of top computing innovators.

The IOI moves from country to country each year, with each host country offering a different unique assortment of activities and cultural excursions during the week-long event. All members of the USA delegation to the 2014 IOI in Taipei, Taiwan, had a wonderful experience, including our four team members, team leader Brian Dean, and deputy team leader and veteran USACO coach Richard Peng. The contest venue was directly adjacent to the Taipei 101 skyscraper—one of the tallest buildings in the world and also, fittingly, a building whose number looks just like “IOI”! During the week of the IOI, we embarked on several cultural excursions, experienced the very best of Taiwanese cuisine, and made fast friends with like-minded peers from all over the world. Combined with our gold medal performance, it was a truly memorable event.

While the USACO is one of the very few organizations in the USA that supports advanced computing at the high-school level, its mission also involves supporting pre-college computing at all levels (a mission of ever-increasing importance, given the disparity between the tremendous demand world-wide for top computing talent compared with the relatively lackluster K-12 infrastructure for teaching computing in the USA). One of our goals is to expand our reach by offering educational materials and contests that can benefit an even wider range of students, and sponsorship from organizations like USENIX is crucial to helping us reach this goal. By continuing to grow our base of participation, we are confident that we will send ever-more talented teams to represent the USA at future IOIs in Kazakhstan (2015), Russia (2016), Iran (2017), and Japan (2018).

For those interested, here is a sample problem (at the silver level) from one of our USACO contests this past season: Suppose you are making a road trip with two navigationally inclined individuals, both of whom are using GPS applications on their phones to help find a good route to the destination. Unfortunately, both applications are using different underlying maps, so they have

differing opinions of the best route to take. Whenever you deviate from the preferred route of one of the GPS applications, it complains loudly that it must recalculate a new route. Given the mapping data for each GPS in a convenient format, your task is to find a route to the destination that results in a minimum number of complaints, collectively, between both GPS units.

For more information about the USA Computing Olympiad, please visit our Web site at <http://www.usaco.org>.

Thanks to Our Volunteers

by Casey Henderson, USENIX Executive Director

As many of our members know, USENIX’s success is attributable to a large number of volunteers, who lend their expertise and support for our conferences, publications, good works, and member services. They work closely with our staff in bringing you the best there is in the fields of systems research and system administration. Many of you have participated on program committees, steering committees, and subcommittees; many have also contributed to this magazine. The rest of the staff and I are most grateful to you all. I would like to make special mention of some people who made particularly significant contributions in 2014.

Although I include them in the list below, I’d like to say an extra special thanks to Niels Provos and Margo Seltzer, who both completed eight years of service on the USENIX Board of Directors this year, reaching the term limit. Both have been dedicated and active Board members for their entire terms, pushing USENIX forward as an organization and giving generously of their time. Following the conclusion of their terms, they both continue to contribute to USENIX as volunteers in various efforts.

Program Chairs

Bianca Schroeder and Eno Thereska: 12th USENIX Conference on File and Storage Technologies (FAST ’14)

Ric Wheeler: 2014 USENIX Research in Linux File and Storage Technologies Summit

Ratul Mahajan and Ion Stoica: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)

Sabrina Farmer, Andrew Fong, and Fernanda Weiden: SREcon14

Dinah McNutt: 2014 USENIX Release Engineering Summit (URES '14) and 2014 USENIX Release Engineering Summit West (URES '14 West)

Chris St. Pierre: 2014 USENIX Configuration Management Summit (UCMS '14)

Jie Liu (General Chair); Sharad Singhal and Bhuvan Ugaonkar (Program Co-Chairs): 9th International Workshop on Feedback Computing

Michael A. Kozuch and Minlan Yu: 6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '14)

Garth Gibson and Nickolai Zeldovich: 2014 USENIX Annual Technical Conference (USENIX ATC '14)

Xiaoyun Zhu (General Chair); Giuliano Casale and Xiaohui (Helen) Gu (Program Co-Chairs): 11th International Conference on Autonomic Computing (ICAC '14)

Steven Swanson: 6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '14)

Jeanna N. Matthews and Sherry Moore: 2014 USENIX Women in Advanced Computing Summit (WiAC '14)

Kevin Fu: 23rd USENIX Security Symposium (USENIX Security '14)

Walter Mebane and Dan S. Wallach: 2014 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE '14); also Editors-in-Chief of the *USENIX Journal of Election Technology and Systems (JETS)*

Chris Kanich and Patrick Lardieri: 7th Workshop on Cyber Security Experimentation and Test (CSET '14)

Jed Crandall and Vern Paxson: 4th USENIX Workshop on Free and Open Communications on the Internet (FOCI '14)

Avi Rubin and Eugene Vasserman: 2014 USENIX Workshop on Health Information Technologies (HealthTech '14)

Zachary N J Peterson: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE '14)

Michael Bailey and Fabian Moore: 2014 USENIX Summit on Hot Topics in Security (HotSec '14)

Sergey Bratus and Felix "FX" Lindner: 8th USENIX Workshop on Offensive Technologies (WOOT '14)

Jason Flinn and Hank Levy: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)

Flavio Junqueira and Keith Marzullo: 10th Workshop on Hot Topics in System Dependability (HotDep '14)

Yuvraj Agarwal and Karthick Rajamani: 6th Workshop on Power-Aware Computing and Systems (HotPower '14)

Ada Gavrilovska and Anthony D. Joseph: 2014 Workshop on Supporting Diversity in Systems Research (Diversity '14)

Ken Birman: 2014 Conference on Timely Results in Operating Systems (TRIOS '14)

Kaoutar El Maghraoui and Gokul Kandiraju: 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW '14)

Nicole Forsgren Velasquez: 28th Large Installation System Administration Conference (LISA14)

Kyrre Begnum and Charles Border: 2014 USENIX Summit for Educators in System Administration (SESA '14); also Chief Editors of the *USENIX Journal of Education in System Administration (JESA)*

Invited Talks/Special Track Chairs

John Strunk: Tutorial Coordinator at FAST

T.S. Eugene Ng and Amar Phanishayee: Poster Session Co-Chairs at NSDI

Jaeyeon Jung: Deputy Program Chair at USENIX Security

Sandy Clark, Matthew Green, Thorsten Holz, Ben Laurie, Damon McCoy, Jon Oberheide, and Patrick Traynor (Chair): Invited Talks Committee at USENIX Security

Franziska Roesner: Poster Session Coordinator at USENIX Security

Allen Clement and Roxana Geambasu: Poster Session Co-Chairs at OSDI

Patrick Cable, Doug Hughes, and Matthew Simmons: Invited Talks Coordinators at LISA

Lee Damon: Lightning Talks Coordinator at LISA

Cory Lueninghoener: Workshops Coordinator at LISA

Tom Limoncelli and Matthew Simmons: Tutorial Coordinators at LISA

Paul Krizak (Chair) and Chris McEniry: LISA Lab Coordinators

Branson Matheson and Brett Thorson: LISA Build Coordinators

Other Major Contributors

Cat Allman, John Arrasjid, David Blank-Edelman, Sasha Fedorova, Daniel V. Klein, Brian Noble, Kurt Opsahl, Niels Provos, Carolyn Rowland, Margo Seltzer, Dan Wallach, and Hakim Weatherspoon for their service on the USENIX Board of Directors

Eric Allman, John Arrasjid, and Niels Provos for serving on the Audit Committee

Brian Noble and Cory Lueninghoener for serving on the Awards Committee

Brian Dean, team leader, and Richard Peng, deputy team leader and veteran coach, for this year's USA Computing Olympiad, co-sponsored by USENIX

Eddie Kohler for his HotCRP submissions and reviewing system

Tadayoshi Kohno for organizing the Work-in-Progress Reports at USENIX Security '14

Jacob Farmer of Cambridge Computer for his sponsorship of the traveling LISA Data Storage Day series and for organizing the Storage Pavilion and Data Storage Day at LISA14

Hugh Brown, Katherine Daniels, and Mark Lamourine for blogging about USENIX and LISA14 activities

Conference Reports

23rd USENIX Security Symposium

August 20–22, 2014, San Diego

Summarized by David Adrian, Qi Alfred Chen, Andrei Costin, Lucas Davi, Kevin P. Dyer, Rik Farrow, Grant Ho, Shouling Ji, Alexandros Kapravelos, Zhigong Li, Brendan Saltaformaggio, Ben Stock, Janos Szurdi, Johanna Ullrich, Venkatanathan Varadarajan, and Michael Zohner

Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Kevin Fu, chair of the symposium, started off with the numbers: a 26% increase in the number of submissions for a total of 350, and 67 papers accepted, a rate of 19%. The large PC was perhaps not large enough, even with over 70 members, and included outside reviewers as well. There were 1340 paper reviews, with 1627 follow-up comments, and Fu had over 8,269 emails involving the symposium. Attendance was strong, with 520 people registered.

Jaeyeon Jung (Microsoft Research) worked as Fu's deputy chair, and will be the chair for the symposium in 2015. Tadayoshi Kohno, past chair, handled both WiPs and the shadow reviewers.

Fu announced the Best Paper award, which went to "Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing," by Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Two Best Student Paper awards went to "DSCRETE: Automatic Rendering of Forensic Information from Memory Images via Application Logic Reuse," by Brendan Saltaformaggio, Zhongshu Gu, Xiangyu Zhang, and Dongyan Xu, and to "Automatically Detecting Vulnerable Websites Before They Turn Malicious," by Kyle Soska and Nicolas Christin.

The Test of Time award, for a paper that was presented at least 10 years ago and is very relevant today, went to Roger Dingledine, Nick Mathewson, and Paul Syverson for "Tor: The Second-Generation Onion Router." Dingledine and Syverson were present and received the award, and also spoke briefly in a short panel after the keynote.

Keynote

Summarized by Rik Farrow (rik@usenix.org)

Phone Phreaks: What We Can Learn from the First Network Hackers

Phil Lapsley, hacker, consultant, entrepreneur, and author of *Exploding the Phone: The Untold Story of the Teenagers and Outlaws Who Hacked Ma Bell*.

Phil Lapsley told us that he spent five years researching and writing his book, including over 500 FOIA requests. He later suggested looking at the Web site for the book, where you can search through his references (explodingthephone.com), something I tried while writing this summary, and I can write that searching worked well for me. And while the book is about phreaks, Bell System hackers, Phil presented the background that made phone freaking not just possible but likely.

AT&T was a US government regulated monopoly, with over 90% of the telephone market in the US. They owned everything, including the phones on desktops and in homes, and, as a monopoly, could and did charge as much as \$5 per minute (in 1950) for a cross-country call. In 2014 dollars, that's nearly \$50 per minute. In the beginning, the system relied on women, sitting in front of switchboards with patch cables in hand, to connect calls. Just like Google today, AT&T realized that this model, relying on human labor, cannot scale with the growth in the use of the telephone: They would have needed one million operators by 1970. Faced with this issue, company researchers and engineers developed electro-mechanical switches to replace operators.

The crossbar switch, first used only for local calls, used the pulse of the old rotary telephone to manipulate a Strowger switch, with each pulse setting the switch to the next position. For long-distance calls, operators would have to find a set of available trunk lines to route each call. AT&T's next step was to build more automated switches, called a 4A crossbar, each so large it filled a city block, eventually building over 175 of them. As these 4A crossbars were networked together, they together comprised the largest machine ever built.

The 4A crossbar used a tone, 2600 hertz, to indicate when a trunk line was idle, and pulses to communicate across the trunk to a distant 4A crossbar. The first "hack" of the system occurred when a blind 13-year-old, Joe Engressia, was whistling along with a song and noticed that it caused a phone call to be disconnected. He experimented by calling information (dialing 411), then whistling again (he had perfect pitch), disconnecting that call as well. With practice, he could "dial" long distance calls just by whistling, and Lapsley displayed an old TV segment of an older Engressia doing just that. And, said Engressia on TV, the phone call was free, too. He had discovered the interface used by the switches, and by operators.

AT&T had a big problem. Fixing the system would be very expensive, so they tried to interest the FBI in prosecuting people using the 2600 Hz tone to make free calls, but the FBI wasn't interested at first. Bookmakers, who arranged bets, made a lot of use of phones, and the ability to make both toll free and unlogged phone calls really appealed to them. By collecting information about bookies using the 2600 Hz tone, the phone company got the FBI's attention.

Generating the 2600 Hz tone, along with the other seven tones used with switching equipment, took a little bit of technical know-how, provided by telephone company documents. The first devices were large, and built into blue boxes, which is where the devices got their name. Steve Jobs and Steve Wozniak learned about blue boxes, and sold them to students living in UC Berkeley dorms. This was their first entrepreneurial enterprise, in 1971,

after learning about the hack via an article in *Esquire* magazine. The phone company countered by building green boxes, devices for detecting the presence of 2600 Hz tones entering switches from locations other than trunk lines or 4A crossbars.

Lapsley ended with a list of observations, such as, if you build it, curious kids will hack it, and later so will organized crime and state actors. Bell Labs can be forgiven for not considering security, because there weren't hackers back then. But we now know better. It's better to build security in from the start, rather than attempting to bolt it on later. AT&T had the 175 gigantic 4A crossbars, plus many thousands of smaller switches, all relying on the 2600 Hz signal. In closing, Lapsley said that if you think things are bad now, just wait: The Internet of Things means that there will soon be trillions of connected devices, all with little to no security designed in.

Vern Paxson praised the talk, then pointed out the mission creep: A search for phreakers turns into an effort to help locate bookies. Lapsley said he had interviewed Bill Caming, a fraud attorney for AT&T, who said the FBI was quite happy to receive a comprehensive list of bookies, and that it was part of an ongoing relationship between AT&T and the government. Paxson went on to say that mission creep was a subset of co-evolution, and Lapsley responded that AT&T was a Stalinish Central Planning organization, but one that doesn't exist any more. Steve Bellovin, speaking as a historical researcher, really appreciated the online references (see above). Bellovin then pointed out that the FBI wasn't interested in cloned cell phones either, until they learned that drug dealers were using them. John Stalwart said he was interested in the special numbers, 000–199, that Lapsley had mentioned. These numbers allowed phone company insiders to test lines, but also to eavesdrop on existing connections, and that because there were so many people employed, it was possible to find someone to bribe.

Tor Panel and Lightning Talks

The summary of this session is available online as an electronic supplement: www.usenix.org/login/dec14.

Privacy Session

Summarized by Kevin P. Dyer (kpdyer@gmail.com)

Privee: An Architecture for Automatically Analyzing Web Privacy Policies

Sebastian Zimmeck and Steven M. Bellovin, Columbia University

Sebastian presented a very relatable problem: Privacy policies are often hard to understand and long to read. Most users simply browse a Web site or click “accept” without appreciating the implications of their actions. What's more, in many jurisdictions, such as the United States, privacy policies are legally binding documents. In response, the authors present Privee, a concept that helps users understand the privacy policies they agree to.

Privee is implemented as a browser extension and has two methods for analyzing policies. First, the browser extension

checks whether a crowd-sourced analysis of a given privacy policy exists in a public repository. If a crowd-sourced analysis of the policy is not available, rule and machine-learning classifiers are used to dynamically analyze and classify the policy. At the end of this work-flow, the results are displayed in a simple UI overlaying the complex policy. The UI reports on six different binary dimensions that are of interest to users, such as: “Does the company encrypt my data when in transit and/or storage?” or “Does the policy allow for collection of personal information?”

A ground truth data set to evaluate the accuracy of Privee was obtained by having experts tag privacy policies. It turns out that the accuracy of the machine-learning-based classification is dependent upon the type of question being asked. As an example, it's typically quite easy to infer the context and meaning of words like “encryption.” However, other words like “disclosure” turn out to be problematic for binary classifiers. Interestingly, both human experts and the Privee extension had more difficulties with ambiguous policy language. Nevertheless, Sebastian was optimistic that there will be future improvement in the classification process. He emphasized that privacy policies tend to become easier to understand and classify over time. This all points to Privee being a very promising approach to empower users.

During the Q&A, someone asked how Privee and its classifiers compared to P3P classifications of privacy policies. Sebastian responded that the P3P privacy policies of Web sites are, in fact, often wrong and, thus, diverge from natural-language policies. However, some of the classifications performed by the Privee extension are comparable to the P3P tags, which hints that the Privee concept might have the same expressive power as P3P.

Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing

Matthew Fredrikson, Eric Lantz, and Somesh Jha, University of Wisconsin—Madison; Simon Lin, Marshfield Clinic Research Foundation; David Page and Thomas Ristenpart, University of Wisconsin—Madison

Awarded Best Paper!

Matthew started by describing an area of medicine that we may not all be familiar with: pharmacogenetics. Pharmacogenetics is the use of patients' health records and genetic information to provide individually tailored drug dosing. For certain drugs like Warfarin, a blood thinner, this is critical—incorrect dosing can kill a patient. For this reason the International Warfarin Pharmacogenetics Consortium (IWPC) publishes a pharmacogenetics-based model that works well for tailoring initial Warfarin dosing to a patient, based on the patient's age, height, weight, race, history, and two specific genotypes.

In the medical community, data sets are rarely published. However, the models derived from data sets are often published. In response, the authors present a novel attack, which they call a model inversion attack. In the case of the IWPC model, the model inversion attack means that given just the model, it's possible to predict the genotype of a specific patient. It is assumed

that the attacker has basic demographics, stable Warfarin dose, black-box access to the model, and marginal priors on patient distribution. Then it turns out that an attacker can use model inversion to infer patients' genotypes based on this data more accurately than guessing based on the priors. The accuracy is nearly optimal, and they provide a detailed argument in the paper.

A natural approach to combating a model inversion attack is to apply differential privacy. Ideally, this would allow us to maximize the accuracy of queries but minimize the chance any specific patients' information could be recovered from the model. However, differential privacy is implemented by adding noise to the underlying data set. As it turns out, this changes the model, which, in turn, changes the initial dosing that a patient would receive. Matthew highlighted that there is now an undesirable tension: When differential privacy is applied, patients are at increased risk of negative outcomes, including mortality, and when differential privacy is not applied, patients' privacy is at risk.

The question and answer session started with a question about diet: There is a complex interaction between Warfarin dosing and a patient's diet. Matthew responded that diet was beyond the scope of this study and not considered in the model. Someone asked what the general feeling in the medical field was with respect to privacy. Matthew, of course, couldn't speak for the medical field in general but said that the physician on this project thought that adding noise to the underlying data set was a bizarre strategy for achieving privacy.

Mimesis Aegis: A Mimicry Privacy Shield—A System's Approach to Data Privacy on Public Cloud

Billy Lau, Simon Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva, Georgia Institute of Technology

Billy started by highlighting the tension between usability and security that many users grapple with. Users want the convenience of applications such as Gmail, Facebook Chat, or WhatsApp but don't necessarily want their private data to be stored in plaintext in the cloud. Mimesis Aegis (M-Aegis) addresses this concern by providing developers a framework for adding end-to-end encryption to applications. However, the key difference benefit of M-Aegis over other solutions is that it doesn't require modifications to or repacking of applications. M-Aegis takes advantage of the accessibility layer that is available on modern mobile operating systems. This is used to create a layer called Layer 7.5 (L-7.5) because it acts as a proxy between Layer 7 (application) and Layer 8 (user) of the OSI network model.

The key challenges in implementing M-Aegis is to ensure user experience is not compromised. Features such as spell check, in-app navigation, and search must be retained, despite the use of end-to-end encryption. As it turns out, using the L-7.5 approach, spell check and in-app navigation are features that were easy to retain on OSes like Android without additional infrastructure. However, search turned out to be slightly more problematic, because server-side cooperation cannot be assumed. To overcome

this issue they adapted a searchable encryption scheme that was presented in a prior work; more details appear in the paper.

Using M-Aegis to implement end-to-end encryption requires per-app engineering to ensure that the proxy layer, L-7.5, correctly captures user input, encrypts it, then relays it to the underlying app. What's more, this additional layer plus encryption requires per-app decisions on how to encode data to ensure that the app correctly accepts the input, because not all apps can handle arbitrary ciphertexts.

Finally, to confirm that M-Aegis does not negatively impact user experience, a user study was performed. It was confirmed that no UI anomalies or performance issues were noticed by the users, despite the new layer of encryption.

Someone asked whether the framework required per-app manual work. Billy said that, yes, per-app engineering is needed, but the framework assists with this process. However, challenges may arise if an app doesn't use native UI rendering. Another questioner asked for clarification of the attack model: Is the attacker assumed to be honest but curious or active? Billy confirmed that an honest but curious adversary was assumed. Someone wondered whether most Android apps use the native UI. Billy confirmed that most Android apps do use the native UI in his experience. Were there any plans to support encrypted images? Billy responded that it's easy to encrypt text, but unless the ciphertext abides by specific formats, it is going to fail when stored. What's more, we need encryption schemes that survive compression. Finally, someone asked who manages keys. Billy replied that M-Aegis has a pluggable key distribution system, and any distribution strategy can be used.

XRay: Enhancing the Web's Transparency with Differential Correlation

Mathias Lécuyer, Guillaume Ducoffe, Francis Lan, Andrei Papancea, Theofilos Petsios, Riley Spahn, Augustin Chaintreau, and Roxana Geambasu, Columbia University

Mathias highlighted the issue of targeted advertising. Companies are aggressive, but not transparent, in how they target users. In some cases, companies may try to determine when you're sick, or they may even try to determine when you're pregnant. However, the ads displayed in response to these detected events don't always make it clear what's being targeted. So Mathias posed the question: Is it possible to construct a generic tool that reveals data misuse? As an example, can we determine which emails sent or received trigger a specific ad displayed to a user?

XRay is the first generic tool that correlates inputs (e.g., emails, Web queries, etc.) to ads output to a user. It turns out that the naive approach of creating many shadow accounts does not scale to address this problem. Therefore, the authors present two novel logarithmic algorithms for correlating user inputs with ads output. One algorithm is simple but not robust. The other algorithm is more complex, but more robust.

The algorithms were evaluated against Amazon, YouTube, and Gmail and do not assume that all ads are targeted. A logarithmic number of shadow accounts for each service were required to perform the evaluation. For Amazon and YouTube, when ads are displayed a reason is given (i.e., “We’ve shown you X because Y.”). This enabled a ground truth data set to determine the classification accuracy. The authors’ algorithms were then applied to identify more subtle advertisements on Gmail through manual labeling. This surfaced interesting results: for example, if someone is having financial problems, they are targeted with ads from subprime lenders.

The question and answer session started with an important question: What should users do with this information? Mathias responded that this is first time they had this information, so they didn’t know yet. An ultimate goal is that they want voluntary transparency from services so that users know how data is being used, and this may lead users to change their behavior. The next questioner asked whether someone shoulder surfing who noticed ads targeted to another user, could exploit information about the user? Mathias responded that this is indeed a real threat, but more transparency is needed. The final three questions concerned the capabilities of the framework. Can it work over time series? Is it possible to have inputs that are more complex than a single word? Did they plan to extend the framework to work across multiple services? The answer to all three questions was yes.

Mass Pwnage

Summarized by Qi Alfred Chen (alfchen@umich.edu)

An Internet-Wide View of Internet-Wide Scanning

Zakir Durumeric, Michael Bailey, and J. Alex Halderman, University of Michigan

Zakir Durumeric first talked about the popularity of Internet-wide scans after releasing ZMap last year, and motivated the work by posing the questions, who is using ZMap and what is the security impact from these fast scanning tools? To answer these questions, the authors collected data from a large darknet during 2013 to 2014, and fingerprinted scanners such as ZMap. In the analysis, they found increasing amounts of scanning activity. A large portion of this activity was targeted at vulnerable services. They also characterized the scanning sources. Surprisingly, they found that instead of botnets, most of the scans came from bullet-proof hosting providers or from China, and also many of them came from academic researchers.

After characterizing the scanning landscape, Durumeric chose three case studies to study the scans related to recent vulnerabilities in Linksys routers, OpenSSL, and NTP. For the Linksys backdoor, scanning activities started within 48 hours, and continued for at least two months. For the NTP DDoS attack vulnerability, nearly all probing traffic was part of large scans and was primarily from bullet-proof hosting providers. For the Heartbleed bug in OpenSSL, scan activity began less than 24 hours after the disclosure, and the volume doubled in the follow-

ing week. The scanning origins were either bullet-proof hosting providers or from China.

Durumeric also talked about the results on the defensive mechanisms for the scans. Although scanning activity largely increased, only 0.05% of the IP space was inaccessible from their scanner, and only 208 organizations have requested exclusion of scanning. They also summarized the scan detection mechanism deployed by organizations. Durumeric concluded that large scans have become common, while the defenders remain slow in responding to these scans.

Following the talk, Bill Cheswick asked whether they would be posting the set of ASes responsible for scanning, as he loves the idea of shunning. Durumeric answered yes, and he added that they are listed in the paper. The session chair, Vern Paxson (UCB), asked how they could tell if scanning was widespread, and Durumeric answered that if they saw a normal distribution, they assumed it was widespread. Paxson then pointed out that some sites block IP addresses for short periods of time, and Durumeric replied that they scanned every day.

On the Feasibility of Large-Scale Infections of iOS Devices

Tielei Wang, Yeongjin Jang, Yizheng Chen, Simon Chung, Billy Lau, and Wenke Lee, Georgia Institute of Technology

Tielei Wang investigated the possibility of infecting iOS devices on a large scale from compromised computers in a botnet. In this work, Wang identified two previously unknown attacks that enable compromised host computers to deliver malwares such as the Jekyll app (presented by him in last year’s USENIX Security) into an iOS device via USB or WiFi-based syncing. Leveraging this vulnerability, a measurement study was then conducted to estimate the population of iOS devices connected with compromised computers in botnets, and they found that 23% of bots in the study can be used to infect iOS devices.

The first attack managed to install Apple-signed malware into iOS devices. A major challenge is that each downloaded app is associated with an Apple ID and cannot run on a device that is bounded with another Apple ID. However, the authors found that this protection can be bypassed when using the iTunes syncing feature. With a man-in-the-middle attack idea, this vulnerability can be exploited from a remote computer’s iTunes, opening the door to iOS devices infected remotely with malwares such as the Jekyll app.

The second attack exploited the device-provisioning process, which is designed for testing purposes. Wang’s work found that provisioning profiles can be installed, enabling the installation or removal of attacker-signed apps from a compromised computer via USB. Besides injecting apps, Wang also found that with a USB connection, a host computer can steal apps’ sensitive files, such as cookies, which are well protected by sandbox-based isolation between apps but can be accessed easily from the host computer.

After demonstrating the threat from host computers to iOS devices, Wang used a measurement study to determine the scale of possible infection from compromised computers in the botnet to iOS devices. To get a lower bound of the percentage of the possible infected iOS devices, Wang analyzed a DNS query data set of a botnet to find out the number of iOS users using iTunes on the compromised computers with Windows systems. The analysis results showed that at least 23.70% of the iOS devices can be infected, and for a large botnet in the data set, the infection range could involve 13 cities.

Yossi Oren (Columbia) asked about a limitation of the attack of injecting attacker-signed apps, which requires an iOS developer license and can only provision 100 iOS devices. Wang replied that the Enterprise iOS Developer License can be used to circumvent that. He was also asked about whether the attack of injecting Apple-signed apps depends on iTunes, and he replied that using iTunes is not necessary since there are other third-party tools that can be used to do syncing.

A Large-Scale Analysis of the Security of Embedded Firmwares

Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti, Eurecom

Andrei Costin motivated the work by talking about many insecure embedded systems such as routers, printers, and cars, and described the challenges of making the analysis of embedded firmware security large scale, including, for example, the heterogeneity of hardware, users, etc.; firmware data set building; and firmware identification and analysis.

To collect firmware data sets on a large scale, the authors used a Web crawler to automatically download files online and set up a Web site for users to submit firmware images. Identifying firmwares requires manual effort and remained a challenge. To unpack firmware images and identify custom formats, they extended BAT (Binary Analysis Toolkit) with a range of additional plugins. To enhance the scalability for unpacking and file carving, which are very CPU-intensive, the system uses a cloud computing platform, where the analysis tasks were distributed to several worker nodes. The analysis on the unpacked firmware images includes correlation, clustering, and data enrichment such as version banners and keywords.

Costin demonstrated their system using some examples. The first example was about the correlation engine for finding similarity between firmware images. He showed that by using fuzzy-hashes, the vulnerability propagation can be studied. The second example concerned private RSA keys stored in the firmware images. Using the RSA key correlation and vulnerability propagation, the private key providing access to the Web interface of some CCTV cameras was found to be reused across many firmware images of the same brand, affecting 30,000 online IP addresses. They also found that the vulnerable components of these CCTV cameras are shared with CCTV cameras from another vendor. Costin concluded with a summary of

their results, which include 38 new vulnerabilities correlated to 140,000 online devices.

Cynthia Irvine (Naval Postgraduate School) asked about whether they worked with vendors to make the firmwares more secure. Costin replied that they had disclosed their findings to vendors and communicated with them. He added that they had a Web site and made their data, including the firmware images and the analysis results, public. Costin was also asked about whether there is any way to characterize the vulnerabilities: for example, to identify whether the vulnerability is from the firmware itself or a third party. He answered that currently identifying firmware is hard to automate and is error prone: for example, finding the version number is usually hard due to diverse firmware file formats. He was then asked about how to judge the coverage of their firmware data set since their sources are online crawling and individual submissions. He replied that currently it is hard to build a representative data set since the embedded systems have a very heterogeneous environment.

Exit from Hell? Reducing the Impact of Amplification DDoS Attacks

Marc Kühner, Thomas Hupperich, Christian Rossow, and Thorsten Holz, Ruhr-University Bochum

Christian Rossow presented work on an Internet-wide study of UDP-based amplification attacks, the authors' attempts to mitigate the attacks, along with discussions on potential next step attacks and root causes. Rossow started by showing the scanning results to study the amplifier landscape. They found that the number of amplifiers remained constant throughout the study, and multiple vulnerabilities simultaneously existed on many systems. They then fingerprinted the devices and found that NTP and SNMP amplifiers largely run on routers, while the majority of NetBIOS amplifiers run on desktop computers. They also found that most protocols had a high rate of amplifier churn, but the NTP protocol only had a negligible rate of churn since NTP amplifiers are usually assigned static IP addresses.

Considering that NTP is the worst among all known vulnerable protocols, they then studied potential mitigations towards NTP amplifiers. Rossow showed a timeline graph to demonstrate their success in the remediation process of notifying the administrators. After releasing articles about the NTP attack, updating the list of potential amplifiers, and weekly notifications, they found the number of NTP amplifiers dropped by 92.4% with an ongoing decrease. Although this campaign was shown to be very effective, they still experienced difficulty in reaching out to the providers.

After showing the influence on the amplifier landscape, they further thought about a potential next step for attackers who may turn to exploiting the TCP protocol for amplification attacks. They studied retransmissions during the TCP protocol handshake and found that the SYN/ACK segments will be sent repeatedly without being stopped by RST responses, thus achieving the effect of overloading the capacity of the victim's

network. Evaluated on HTTP, Telnet, and CUPS, this amplification attack was found to enable an amplification factor of six or higher for most of the reachable hosts.

Finally, they studied the root cause for amplification attacks: IP address spoofing. A remote spoofer test was designed based on DNS, without the need for individuals running manual or tool-based tests. Using this remote test method, they identified more spoofing-enabled ASes than previous work.

Cynthia Irvine asked about their future plans. Rossow answered that they want to focus on identifying the source of the amplification traffic, even though it is spoofed. Someone asked whether they used ZMap in their Internet-wide scanning of UDP-based amplifiers. Rossow answered no and added that they developed their own scanning tool. Someone asked which TCP protocols could be exploited for amplification attacks, and Rossow answered that, for example, HTTP Telnet, and FTP can be exploited. The questioner asked whether these TCP-based amplification attacks were real. Rossow replied that they were real and reproducible.

Privacy Enhancing Technology

Summarized by David Adrian (davadria@umich.edu)

Never Been KIST: Tor's Congestion Management Blossoms with Kernel-Informed Socket Transport

Rob Jansen, US Naval Research Laboratory; John Geddes, University of Minnesota; Chris Wacek and Micah Sherr, Georgetown University; Paul Syverson, US Naval Research Laboratory

There is a large body of work showing that Tor is slow, but little work has been done to explain why or to locate where the congestion occurs in the Tor network and client. Rob Jansen presented work that measures where congestion occurs both in the client and in the network in order to determine and help eliminate the root causes of congestion.

By instrumenting the Tor client, the authors were able to measure the amount of delay that Tor packets spent in kernel-level and application-level buffers. After enhancing the Shadow Tor network-simulator to provide more TCP-level data and creating a simulated network of over 3000 Tor nodes, the largest simulated network to date, they were able to determine that congestion almost exclusively occurs in outbound kernel buffers.

To reduce the congestion, the authors reimplemented the Tor client to more efficiently schedule circuits across multiple sockets, and to keep Tor packets in application-level buffers in cases where the send call would cause the packet to be added to a queue rather than being flushed to the wire. By keeping the packet in Tor level-buffers, Tor is able to make more educated decisions about which socket to use. Jansen referred to their improvements as KIST (Kernel-Informed Socket Transport).

KIST resulted in less kernel-level congestion and more Tor-level congestion. However, this produced a net decrease in latency. This latency decrease does make certain attacks published by Hopper

et al. more effective, but the authors believe this is purely due to the fact that latency is lower and is not a direct flaw of KIST.

Someone asked whether they had measured what happens when some Tor clients have the authors' changes and some do not. Jansen stated they had not yet explored this area, and possibly will attempt to measure this before KIST is fully rolled out.

Effective Attacks and Provable Defenses for Website Fingerprinting

Tao Wang, University of Waterloo; Xiang Cai, Rishab Nithyanand, and Rob Johnson, Stony Brook University; Ian Goldberg, University of Waterloo

Tao Wang explained that the goal of Web site fingerprinting is to determine from Tor network flows which Web sites a particular Tor user is visiting by fingerprinting the traffic of a specific Web site. Wang presented a new, more effective fingerprinting attack and a provable defense against it. However, the provable defense has a large performance impact. To explain the work, Wang used comical slides featuring a small fuzzy character who was using Tor and being attacked by a red blob.

The attack is machine-learning-based and uses the k-nearest neighbors algorithm with a specially trained distance measurement that weights various features visible in encrypted Tor flows, such as packet size and timing. With the classifier trained for a variety of popular Web sites, when used against a Tor client visiting both fingerprinted and non-fingerprinted Web sites, the attack is able to achieve an 85% true positive rate with a 0.6% false positive rate.

To defend against the attack, the authors implemented a provably secure defense that uses the shortest-common super-sequence to enforce that all packet flows appear identical. However, while this method is provably secure, it requires at least a 60% bandwidth overhead. A non-provably-secure defense can use as little as 6% bandwidth overhead, but can still be broken with enough data, effort, and time.

TapDance: End-to-Middle Anticensorship without Flow Blocking

Eric Wustrow, Colleen M. Swanson, and J. Alex Halderman, University of Michigan

Eric Wustrow explained that since 2011, there have been several different papers that describe methods for performing anticensorship using end-to-middle proxies, such as Decoy Routing, Telex, and Cirripede. However, when the authors of this paper and Telex approached ISPs to deploy these end-to-middle proxies, the ISPs opposed the inline flow-blocking element—the part of the proxy that blocked traffic to the decoy host and redirected it to the censored host.

TapDance is an end-to-middle proxy that only needs a passive tap and the ability to inject packets, in place of the flow-blocking element. To achieve this, the authors used an insight gained from a careful reading of the HTTP and TCP specification. By sending an incomplete HTTP request to the decoy server that is

missing the second `\r\n`, the decoy server will ignore the rest of the incoming packets that are intended for the censored host.

To indicate to the TapDance station that the incomplete request is in fact a TapDance request, the authors use a steganographic channel inside of the TLS ciphertext. For stream ciphers, the authors leverage the fact that flipping a bit on the input flips the corresponding bit on the output. Using this channel, the authors can encode a message encrypted with a TapDance station's public key.

Once the connection has been established, the censored client can send requests for censored content to the decoy server that are then decrypted by the TapDance station and proxied via an uncensored network to the rest of the Internet. The TapDance station then spoofs a response from the decoy server with the censored content.

Peter Honeyman expressed his surprise that any ISPs were willing to deploy such an anti-censorship tool, with or without flow-blocking. Wustrow replied that the amenable ISPs tended to be Tier 2 ISPs with a research focus, rather than a large consumer ISP such as Comcast.

A Bayesian Approach to Privacy Enforcement in Smartphones

Omer Tripp, IBM Research USA; Julia Rubin, IBM Research Israel

Omer Tripp described a tool designed to detect privacy leaks in non-malicious Android applications, such as applications that send user data to third-party advertising companies. Their approach uses a Bayesian classifier and assumes data is transferred using any of several common encodings, such as hex, base64, and JSON.

The authors implemented their system in a tool called Bayes-Droid. When applied to 54 applications from Google Play, the authors were able to find 27 new privacy violations with only one false positive. These results were more accurate with fewer false positives than TaintDroid, the current state-of-the-art taint tracking tool for Android privacy analysis.

Someone asked whether their small sample size might have led to overfitting. Tripp replied that they did not believe so, and that they were releasing their code with the hopes that third parties might use it and provide feedback and more training data.

Crime and Pun.../Measurement

Summarized by Andrei Costin (costin@eurecom.fr)

The Long "Taile" of Typosquatting Domain Names

Janos Szurdi, Carnegie Mellon University; Balazs Kocso and Gabor Cseh, Budapest University of Technology and Economics; Jonathan Spring, Carnegie Mellon University; Mark Felegyhazi, Budapest University of Technology and Economics; Chris Kanich, University of Illinois at Chicago

Janos Szurdi started his talk by introducing "typosquatting" domains. These are domain names that differ from domain names of known or established brands by a small difference: for example, "google.com" versus "googl.com". Users usually end

up on these domains by having a typo in the intended domain names, hence the term "typosquatting." Janos mentioned there are around 56 million potential typosquatting domains in the whole .com range. And while most previous research focused at most on the top 200,000 Alexa sites to look for typosquatting intelligence, they performed a comprehensive study across the entire .com domain distribution to gather a more complete understanding of the typosquatting phenomenon.

Janos presented a case-study example based on PNCBank, which is a Top 10 US bank and is hosted at pncbank.com. There exists pncbnk.com, which runs advertising, pncban.com, which runs a survey that turns out to ask for credit card and personal details at the end, and, finally, pvbank.com, which claims to be a bank in India and even has a disclaimer that states the site is not a typosquatting domain! The case of pncbank.com, Janos says, could have been missed by previous researchers due to its lower Alexa ranking. Moreover, only 2% of the typosquatted domains were found to be associated with their related brand owners (either by domain owner directly or via brand protection services). The other 98% of the cases were parked serving ads, and a minority of them were associated with either competitors' domains, phishing, or malicious pages.

In their methodology, Janos said, they used 1-Levenshtein distance to generate typosquatting based on real and valid domains. Thus they used: deletion of a character, addition of a character, substitution of a character, switching of two adjacent characters, and appending the "www" prefix to the TLD domain name.

Janos noted that some mistyped domains are "true typosquatting" domain cases, while some are not and are "incidentally typosquatting" (i.e., not a true typosquatting case) similar to the original brand only incidentally. To find a "true typosquatting" case, several pieces of information are used, such as DNS and WHOIS records, the domain's content, and redirection chains. However, there was a need to develop heuristic rules to decide whether a domain is an instance of "true typosquatting" or "incidental typosquatting." The challenge in developing such heuristics, however, is the lack of ground truth.

Based on Alexa, the authors took four sets of .com domains to form the ground truth. Sets were as follows: Alexa rank 1–10,000, Alexa rank 10,000–250,000, Alexa rank 250,000–1,000,000, and a set of randomly chosen .com domains. For each set, they performed typo-generation (as presented above) and selected 100 random domains from each typo-generated set. Finally, they compared their results with the other two previous works. For the top domains set, they found that most typo domains are "true typosquatting" domains, and all three classifiers performed with similar accuracy. For the other three less popular domain sets, the other two classifiers decreased in performance, while Janos' classifier kept almost the same performance as for the top domains set. In addition, for the set Alexa rank 250,000–1,000,000, and the set of randomly chosen .com domains, less than 70% were found to be typo-domains.

Janos highlighted the following trends in the registration time of typosquatting domains. There is a continuous battle between brand protections and typosquatters. Obviously, over time typosquatters cannot register popular domains. In addition to this, more than 70% of typosquatting domains are registered more than one year, which means domains are truly desired/desirable.

Janos pointed out that 25% of typosquatting domains were registered with one registrar, and another 15% with six registrars. One solution, therefore, would be for ICANN and VeriSign to force on such registrars additional checks to prevent typosquatting. However, there is no incentive to do so since it will decrease the revenue of both ICANN, VeriSign, and the registrars.

Janos concluded that the typosquatting phenomenon is widespread and is also targeting less-popular domains as well. Another conclusion drawn was that the number of popular typosquatting domains increases over time.

Jeffrey Goldberg (AgileBits) asked whether some substitutions are more common than others for genuine typosquatting. Janos mentioned that deletion and so-called “fat finger” typos were the most common substitutions. Someone asked whether one big group was behind the typosquatting, or hundreds of groups, and what indicators could and would be used to answer such a question. Janos clarified that there are few groups massively typosquatting, as well as some other smaller groups for whom typosquatting was not the main business. As for differentiating between the groups, authors used DNS, WHOIS, and contents of the site. Janos was asked whether typosquatting domains use HTTPS and whether HTTPS is more or less common among the typosquatting domains. Janos said they unfortunately didn’t look at HTTPS and this is a good point to look at in the future. The final question was whether the auto-correction features (e.g., in browsers and search engines) would help to avoid landing on typosquatting domains. Janos said they didn’t study this direction, but he thought it would help a lot.

Understanding the Dark Side of Domain Parking

Sumayah Alrwais, Indiana University Bloomington and King Saud University; Kan Yuan, Indiana University Bloomington; Eihal Alowaisheq, Indiana University Bloomington and King Saud University; Zhou Li, Indiana University Bloomington and RSA Laboratories; XiaoFeng Wang, Indiana University Bloomington

Sumayah Alrwais began by introducing “parked domains.” These are “unused” domains that receive a large amount of traffic and that usually register with advertising networks for high revenues. A common choice for running such domains are domain parking services (DPS). These are usually running various advertising network campaigns and are addressed to various markets, advertising types, and revenues.

Sumayah presented several examples depicting different parked domain scenarios. One is “education-guide.org,” registered with a DPS and running multiple advertising networks (Google AdSense, advertise.com, Bing Ads). Another is “city-cars.net,” which sells traffic for specific search keywords. This type

of traffic is sold via keyword-related companies and traffic systems such as DNTX.com. A final example is “expeedeaa.com,” which is an obvious typo-domain for “expedia.com” and is registered with a brand protection service.

Sumayah briefly explained the ways the domains are parked and monetized. One option is to simply redirect traffic via HTTP 302 redirects. Another option, the most common one, is to set the name servers (NS) of the domain to those of the DPS, such as ns1.sedo.com. In this case, it gives the DPS complete control over the traffic, hence the best content, monetization, and redirection strategies.

Sumayah explained that they wanted to study the legitimacy of the DPS operation. Are they honestly reporting revenues to all the parties using their services? Are the advertisers receiving real clicks, real traffic for purchased keywords, and are the landing pages safe and free of malware? She explained there are inherent difficulties while investigating DPS. One challenge is the complex system of DNS and registrations, in addition to the many parties involved (DPS, Web users, domain owners, advertisers, ads-networks, proxies). Another challenge is that every actor has only a partial view of the entire monetization chain.

Sumayah presented their approach, which basically is an attempt to capture the end-to-end view of the monetization chain by using multiple actors to enter the domain parking monetization ecosystem. One actor registered domains and parked them with DPS (as domain owner). Another set up sites for “fake” advertisers (as advertisers) and bought advertising and traffic keywords. Finally, as Web users, they instrumented Web crawlers to search for millions of parked domains. Some of the crawled domains were operated by the authors themselves. She explained that this activity proved non-trivial with many challenges. From the advertiser point of view, they had to impersonate multiple user-agents of Web users. As traffic purchasers, they had to purchase keywords similar to domain names owned by the authors. The whole experiment consisted of capturing and analyzing 1.2 million chains, with 1015 end-to-end seed chains used as ground truth. The authors then applied fingerprinting on the monetization chains by using click/traffic stamps inside URL patterns and IP addresses that identified a particular monetization party.

Sumayah described the many frauds they discovered. The authors found click fraud in 45.7% of chains and at least 709 fraudulent clicks. In 38% of chains, they also detected “traffic spam” fraud, by receiving purchased traffic from completely unrelated domain names (e.g., for keyword “coupon”). They found malware distribution (via social engineering or drive-by downloads) in 2% of the chains. Finally, they detected “traffic stealing” fraud, which means not reporting the whole revenue to the domain owners. In this particular case, the authors paid 10 cents as traffic buyers. However, the assumed revenue did not propagate to the parking domain provider or to the real domain owner (who parked the domain, the authors in this experiment).

This resulted in the fact that traffic buyer got fully billed while the real domain owner got no revenue at all.

In conclusion, Sumayah said they estimated that more than 40% of the revenue of PDS is illicit. This, in the authors' view, calls for immediate regulation of domain parking businesses. Until then, there are several mitigations possible. One is that search advertising networks should label publishers with categories. Another is the integrity check and protection on the traffic buyers' side.

Andrei Costin (Eurecom) asked why Google entered the DPS market and why it left shortly thereafter. Sumayah thinks Google entered the market to tap into the search-keywords pool. However, Google got sued and most probably it was a big legal pain, which meant that it was easier to exit the market than to deal with all the legal and unregulated aspects. A follow-up question by Andrei was whether Sumayah could provide some advice regarding taxes, IRS, and legal aspects of their research. Sumayah encouraged researchers to always speak first to their legal department as this would surely solve many of the issues upfront, at least it did for them. Someone asked about the exact mechanism DPS created for click frauds. Sumayah said the fraudulent DPS had no hidden iFrames but were simply redirecting the click to a referral link. This resulted in a valid click from the advertiser point of view. Someone asked whether they reported the frauds to the FTC, and if so was the FTC interested in the reports. Sumayah said they did not report their results to the FTC.

Towards Detecting Anomalous User Behavior in Online Social Networks

Bimal Viswanath, M. Ahmad Bashir, Max Planck Institute for Software Systems (MPI-SWS); Mark Crovella, Boston University; Saikat Guha, Microsoft Research; Krishna P. Gummadi, Max Planck Institute for Software Systems (MPI-SWS); Balachander Krishnamurthy, AT&T Labs-Research; Alan Mislove, Northeastern University

Bimal Viswanath started by presenting service abuse as an anomalous user behavior problem. Service abuse is a serious problem today, where, for example, for less than \$20 one can buy 5,000 "Likes." This phenomenon has especially negative effects for social advertising services.

The goal of Bimal's team is to detect and limit service abuse. This includes detecting the identity of such users and nullifying their identities or accounts. This is challenging, however, because of the adversarial cycle, where the "attacker" (i.e., abuser) mutates and always has an upper hand over the system. It is well known that attackers mutate by using fake accounts or compromised real accounts, or when real users collude with their identities to boost their ranking.

Bimal and his team suggested the approach of building an anomaly classifier. It is completely unsupervised and requires no training or labeled data. Additionally, it requires knowledge of attackers' strategies. Their approach contributes to the detection of "Like" spammers on social networks such as Facebook, regardless of spammers' strategies. Bimal indicated that for the approach to work, the classifier needs to learn patterns of nor-

mal user behavior. Hence, to evade detection the attackers need to act along with this learned user behavior, which limits and constrains the attackers in their actions.

Bimal introduced tables of "Normal vs. Anomalous" users. These are based not just on the number of categories and number of likes in each category, but also on an "inconsistent" small number of categories. He noted that behavior also changes over time, so they used PCA (Principal Component Analysis) to detect anomaly. Then he presented the capture of normal behavior patterns. To detect the few patterns of behavior that are dominant, Bimal said they used variance captured by each principal component from PCA. He confirmed this approach to work on Facebook, Yelp, and Twitter.

Bimal explained they trained the classifier on the behavior of "Like" activities of a random large sample of Facebook users. Subsequently, they tested the classifier on 3,200 black market accounts (bots, fake accounts), on 1,000 compromised accounts, on 900 colluding accounts, and on 1,200 normal accounts. Their classifier successfully flagged 99% of black-market accounts. He then explained "click spam" detection on Facebook. They set up a real advertisement targeting US users for a survey link. They also set up a bluff advertisement (i.e., an almost empty ad) that would be expected to get almost no clicks. The surprising result he mentioned was that both ads received a similar number of clicks and similar activity on the landing page!

Bimal concluded that service abuse is a huge problem in social networks today. He and his team proposed an unsupervised anomaly detection scheme. They then evaluated their technique on extensive ground-truth data of anomalous behavior. Finally, they applied their approach to detect click-spam in a social network advertising platform.

Damon McCoy (George-Mason) asked whether the technique's applicability domain is highly constrained by the patterns analyzed. Bimal replied that the technique is quite general and works as long as one can model the "normal behavior" of a user in a given system. Someone asked whether the adversaries they assumed have specialized in compromised accounts. Bimal explained that it would be hard to catch compromised users, but the behavior of such accounts after compromise makes them easier to catch with the proposed system. A follow-up question was on the number of "normal users" to be compromised in order to manipulate the technique. Bimal suggested that it would require compromising more than 30% of the "normal users." Algis Rudy (Google) asked about the threshold for noticing the anomalous behavior. Bimal explained that it largely depends on the operator, but it is a tunable parameter that can be trained for false-negative and false-positive ratios, and also depends on how much threshold the operator wants to tolerate.

Man vs. Machine: Practical Adversarial Detection of Malicious Crowdsourcing Workers

Gang Wang, University of California, Santa Barbara; Tianyi Wang, University of California, Santa Barbara, and Tsinghua University; Haitao Zheng and Ben Y. Zhao, University of California, Santa Barbara

Gang Wang began the talk by introducing machine learning (ML) in the context of security and detection of malicious users. ML is a proven tool for security applications. It's widely used for email spam detection, intrusion and malware detection, authentication, and identification of fraudulent accounts (Sybils). Despite its success, ML also has weaknesses. The key vulnerability is that statistical classes are derived from a fixed data set. Strong adversaries may become aware of ML and try to circumvent it.

Gang's team focused on crowdturfing, defined as malicious crowdsourcing, which is the act of hiring a large army of real users for malicious attacks or activities. For example, it is used to create fake reviews, fake testimonials, political campaigns, and CAPTCHA solving. An online crowdturfing system focuses on finding crowd workers for a customer. In China, ZBJ and SDH are the two biggest such systems with revenues of millions of USD per year.

Gang suggested that ML can be used against crowdturfing. One reason is that it can do more sophisticated modeling of user behavior. Another is that it's the perfect context in which to study adversarial ML, where there are highly adaptive users seeking evasion. In the case of adversarial ML, he suggested the existence of "evasion attacks" (i.e., workers/attackers evade classifiers) and "poison attacks" (i.e., workers/attackers alter training data).

Gang then presented their goal to develop defenses against crowdturfing targeting Weibo. Other goals are to understand the impact of adversarial countermeasures, that is, to understand which classifiers are more accurate than others and in which scenarios one classifier outperforms the others.

Their methodology was to gather training data and build/train classifiers afterwards. As their ground-truth data set, they used the two largest crowdturfing services targeting Weibo (ZBJ, SDH), totaling three years of data with more than 20,000 Weibo campaigns. In addition, they used 35 features to train the classifiers for crowdturfing.

The performance of 60% for random forests (RF) and 50% for decision trees (DT) indicates that it's possible to build an accurate classifier to detect crowdturfing workers. The follow-up research challenge was to detect workers trying to evade the classifier by mimicking a "normal user" and also to understand what knowledge is practically available to evaders.

Gang presented the set of evasion models they designed. One is "optimal evasion," which can be per-worker optimal or globally optimal. Another is the "practical evasion scenario," where the attacker does not know the classifier threshold boundary,

but only knows estimated normal user statistics and can adopt those features that make an action look "normal." For optimal evasion attacks, results showed that 99% of workers could evade the classifier by changing five or fewer features. Although the practical evasion attack required more features to be changed, most classifiers were found vulnerable.

Gang then discussed the poisoning attack, which is executed during the classifier training phase by crowdturfing service admins highly motivated to protect their Web sites and workers. These attacks use tampering of the training data of normal users to manipulate model training. One way to do this is to inject mislabeled samples to training data, which results in a wrong classifier. Another way is to alter worker behavior uniformly by enforcing system policies, hence making it hard for the classifier to separate those two subsets. For example, admins can force a predefined time-out for workers before taking another task, so by delaying workers' tasks the service admin preserves the worker. Gang underlined the effectiveness of poisoning attacks, especially where more accurate classifiers are more vulnerable. For example, 10% of poisoned samples boost false positives by 5%.

Gang concluded with some key observations. One is that accurate ML classifiers can be highly vulnerable. Another is that no single classifier excels in all the attack scenarios. As future work, Gang mentioned the improvement of the ML classifiers' robustness.

Someone asked whether selection of the attributes/features is important for building the model. Gang explained that in their experience all features are useful, but some are easier to modify. He mentioned that it would be interesting in the future to look at features versus the cost of their modification. Kurt Thomas (Google) mentioned last year's semi-supervised study and asked which was better for detecting crowdturfing. Gang explained that last-year's semi-supervised approach was mainly aimed at detecting Sybil attacks in social networks and assumed that the baseline majority of users were stable, good-faith users. For crowdturfing, in particular, the adversaries have changed: They are real users, and it's easier for them to adapt, and hence they require supervised learning. However, unsupervised learning for crowdturfing looks very prospective as well.

Work-in-Progress Reports

The summary of this session is available online as an electronic supplement: www.usenix.org/login/dec14.

Forensics

Summarized by Grant Ho (grantho14@cs.stanford.edu)

DSCRETE: Automatic Rendering of Forensic Information from Memory Images via Application Logic Reuse

Brendan Saltaformaggio, Zhongshu Gu, Xiangyu Zhang, and Dongyan Xu, Purdue University

Awarded Best Student Paper!

Brendan Saltaformaggio presented work on automatically reconstructing file content from memory images (snapshots of a system's volatile memory). While existing forensics techniques can recover many of the raw data structures from memory images via signature scanning, significant work still remains in order to interpret or extract meaningful content from the raw data structures; Saltaformaggio dubs this the “data structure content reverse engineering” problem, and the core contribution of this paper is a technique and system that tackles this problem. Their main idea is that the applications where data structures are found often contain the rendering and formatting logic needed to generate human-understandable content from the data structures; thus, DSCRETE attempts to identify and reuse these rendering/formatting functions (collectively referred to as the “P function”) in a program's binary to automatically extract human-understandable output from memory images.

First, an investigator feeds an application binary (e.g., the PDF viewer on the machine under analysis) to DSCRETE. Next, DSCRETE automatically discovers rendering/printing functions in the binary by using slicing. A list of candidate entry points to the P function (the first function that will call the correct rendering functions to generate understandable content) is then created by searching for locations that take a heap pointer and for which all previously discovered rendering functions depend on (based on a dependency graph). The final, correct entry point is then found through “cross-state execution”: repeatedly feeding data structures from the memory image to candidate entry points until a human-understandable file is generated; this correct entry point can be saved for future analysis.

Saltaformaggio concluded by noting that DSCRETE was highly effective at recovering many digital documents and presented several demos of DSCRETE recovering Gnome-paint images and PDF documents from memory images.

Thurston Dang (UC Berkeley) asked whether DSCRETE can handle files generated by interpreted applications or scripts. Saltaformaggio said that, currently, DSCRETE cannot handle such files, but he affirmed that this would be pursued in future work. Responding to another capabilities question from a researcher from the University of British Columbia, Saltaformaggio remarked that as long as ASLR was disabled on the investigator's machine, the memory image could come from a machine that used ASLR because of memory mapping techniques discussed in more detail in their paper. Finally, David Jacobson (Qualcomm) asked whether DSCRETE could handle memory

images that were partially corrupt; Saltaformaggio replied that DSCRETE should perform fine so long as the kernel paging structures were intact, but even if they were corrupt, techniques such as DIMSUM from NDSS 2012 might be able to be used in conjunction with DSCRETE to effectively generate human-understandable content.

Cardinal Pill Testing of System Virtual Machines

Hao Shi, Abdulla Alwabel, and Jelena Mirkovic, USC Information Sciences Institute (ISI)

Hao Shi presented his work on systematically discovering red pills that can detect whether a program is being executed in a VM or real/bare-metal machine; this was joint work with fellow researchers at the University of Southern California. Although several existing papers discuss how to build red pills, Shi's work is the first to systematically explore the entire space of the Intel instruction manual to detect semantic and computation differences between a VM and bare-metal machine.

Rather than use randomized testing, which lacks complete coverage, or symbolic execution, which cannot assess computational differences such as floating point arithmetic, Shi's group generated a red pill test case for all instructions in the Intel x86 manual. They then executed each of these red pill test cases (instruction operations) on a bare-metal machine, Bochs, QEMU using TCG, and QEMU using hardware-assisted virtualization; after executing the red pill tests, they compared the system states of their bare-metal machine against each of the VMs to determine whether the red pill elicited a difference between the VM and bare-metal machine. For each VM, Shi's work discovered over 7,000 red pills that could be used to distinguish the VM from a real-machine.

Concluding his presentation, Shi noted that their red pills emerged from two sources: incorrect VM implementations of the instructions or under-specified instructions that result in naturally ambiguous and different implementations of x86 instructions. Looking forward, Shi sketched a defense idea against red pills that instruments a honeypot/VM so that it takes in the list of red pills discovered by Shi and automatically executes/returns the correct and expected value of a real machine.

One researcher noted that Shi's presentation and paper mentioned that they tested several different versions of QEMU such as 1.3.1, 1.6.2, and 1.7.0, but neither the presentation nor paper included results for the later versions of QEMU (1.6.2 or 1.7.0); Shi affirmed that his paper did contain results for 1.6.2 and 1.7.0, despite challenges from the audience member that the results could not be found in the published, online version. Since Shi's group tested for red pills across different versions (i.e., newer versions) of QEMU, one audience member asked whether it looked like QEMU was improving the fidelity of its emulation of a real machine; sadly, Shi noted that based on their test results, it does not appear that QEMU is improving.

BareCloud: Bare-Metal Analysis-Based Evasive Malware Detection

Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel, University of California, Santa Barbara

Dhilung Kirat presented work on detecting binaries that attempt to evade VM analysis. While many dynamic analysis techniques have been developed to analyze malware samples, these analysis techniques are often conducted inside of a virtual machine called a “honeypot” when used for malware analysis. Unfortunately, a number of techniques have been developed that allow binaries to detect whether their execution environment is a VM or real (bare-metal) machine. The major contribution of Kirat’s work is “BareCloud,” a bare-metal analysis system and a technique to leverage the bare-metal system’s analysis to detect whether a binary sample exhibits cloaking/evasive behavior in a VM.

BareCloud sets up the bare-metal machine with a clean system snapshot and then initiates a binary sample by sending it over the network and removing all traces of in-guest automation and analysis tools before the malware sample executes. After allowing the sample to execute, BareCloud extracts a behavioral profile from the bare-metal machine, which is a tree-based structure of the disk changes (relative to the clean system snapshot) and network activity; in order to keep the bare-metal system completely free of in-guest, detectable monitoring tools, the information collected by the bare-metal analysis is limited to just disk and network activity. Next, BareCloud extracts similar behavioral profiles from common honeypots by loading the same clean snapshot onto Anubis, Ether, and Cuckoo Sandbox and executing the same binary sample in those environments. These tree-structured behavior profiles are then compared using hierarchical similarity to detect significant deviations (i.e., evasion) between the bare-metal machine and honeypots; a significant deviation is defined by an empirically derived threshold set by the researchers.

To evaluate BareCloud, Kirat’s team ran BareCloud on a corpus of 110,000 samples from Anubis; the samples varied from minimal activity (very few system events and no network traffic) to high activity (thousands of system events and more than 10 packets set over the network). Overall, they found that BareCloud labeled approximately 6,000 samples as evasive malware.

Inquiring about these results, Grant Ho (UC Berkeley) asked whether Kirat’s team had analyzed what the false-positive and false-negative rates were for their 6,000/110,000 result metric. Kirat responded that there was no ground truth labeling, so it wasn’t possible to precisely determine how many of the 6,000 samples might be false positives and how many evasive malware samples remained undetected in the full set of samples. Additionally, Kirat was asked whether the execution environments (i.e., the operating system version, libraries, and applications like Java) were identical among all sandboxes and the bare-metal machine. Kirat said that during the BareCloud experiments, all execution environments were kept exactly the same to ensure

that behavior profile differences truly resulted from a sample’s behavior. Answering another question about identical environments, Kirat noted that Anubis’s auto-click feature (which allows the honeypot to automatically interact with samples) was disabled because the bare-metal system cannot have the same auto-click behavior without introducing detectable, in-guest automation components.

Blanket Execution: Dynamic Similarity Testing of Program Binaries and Components

Manuel Egele, Maverick Woo, Peter Chapman, and David Brumley, Carnegie Mellon University

Manuel Egele presented his work on a dynamic analysis technique to determine whether two functions are similar given just their binary representation. Determining whether two functions are actually similar or different, given their binary forms, has several security applications such as identifying polymorphic malware or analyzing updates/patches for vulnerability findings. Their tool, “Blex,” introduces a novel dynamic analysis execution technique (dubbed “blanket execution”), which greatly increases coverage of program logic.

Like many existing dynamic equivalence checkers, Blex starts by executing two unknown functions, f and g , in a fixed environment and storing the side effects of both executions. But in addition to their initial execution, the blanket execution process continues to execute f and g (and collect their side effects) by finding the first unexecuted instruction in f and g and executing them from that point; this repeated execution and side effect collection continues until all instructions in f and g have been executed at least once. From the full set of side effects for f and g , Blex outputs a similarity score for the two functions by computing a Jaccard index of the two sets.

To evaluate the efficacy of Blex, Egele’s group collected the functions from the GNU coreutils package and ran Blex on these functions, when compiled at different optimization levels. While BinDiff slightly outperforms Blex when functions are compiled at similar optimization levels (e.g., $-O0$ vs. $-O1$), Blex is roughly twice as effective as BinDiff when matching functions at very different optimization levels (e.g., $-O0$ vs. $-O3$).

Egele was asked how Blex performed when functions and binaries were obfuscated, as typically seen in malware. Egele responded that enhancing Blex to handle obfuscated code will be future work; this work was simply the first step to see whether their blanket execution technique was even possible, which is why they evaluated Blex on obfuscated versions of popular binaries.

Invited Talk

Summarized by Rik Farrow (rik@usenix.org)

Information Security War Room

Sergey Bratus, Dartmouth; Felix (FX) Lindner, Recurity Labs

Sergey and FX filled in at very short notice for a speaker unable to attend and provided a lively presentation. Unfortunately, FX was in DEFCON mode, and much of the talk was R-rated for language. Nevertheless, what the two had to say was definitely interesting and relevant to our current security landscape.

The speakers really had four main points to make: (1) we do not currently have secure systems; (2) there is no product liability for the software industry; (3) instead of fixing the problem, some countries are trying to defend the status quo using treaties; and (4) LangSec principles explain a lot of the security issues in current software.

FX began by pointing out that you cannot buy a secure system today, and that just two market sectors have no product liability: software and illegal drugs. Both of these groups call their customers “users,” a comparison that earned FX some laughs. FX then displayed a graph created by Veracode, a company that examines software binaries looking for exploitable input conditions. The two worst software sectors for security flaws were in customer support software and security software. This should not be a surprise since security software usually has the focus of being built out of protocol parsers, a topic they get to later. But just imagine, said FX, having a Intrusion Prevention System with all of its rules built into the operating system: you have 500 protocol parsers right in the line of sight of the attacker.

FX, who acts as a NATO advisor in the cybersecurity realm, said he has hope that the military will be the first sector that requires that software be backed with product liability guarantees. FX said that the military already takes eight years to procure software, and buggy and insecure software empires could receive a fatal blow if companies will not stand behind the software that they sell by accepting liability for failures.

FX shifted into a related theme, the attempt to create treaties to “fix” the software security problem. He used the analogy of long bows, which made it possible for peasants to kill the expensively armored nobility and resulted in said nobility creating rules of warfare in an attempt to outlaw this “unfairness.” FX described a simply amazing example, the new code of conduct for the Internet, proposed by China, Russia, Tajikistan, and Uzbekistan (and other countries) at the 66th session of the United Nations. FX claimed that this is not because Russia and China feel threatened by cyberattacks, but instead they feel they are in a superior position.

But FX and Sergey feel that the Wassenaar Arrangement is a much bigger threat to security today. This treaty even makes research into tools that might be used in cyberwarfare illegal. For example, software designed to avoid detection by monitor-

ing tools, for extraction of data or information, or to modify the standard execution path of a program, would be illegal to export. As an example, Sergey pointed out that debuggers and dynamic loaders both modify the execution path, although exceptions are made for these by Wassenaar. But tools like debuggers came out of a need to understand how software works (or doesn't work), and treaties like this one would ban research into potentially dual-use tools. The speakers, and several others, have written a position paper and call to action that can be found in the online version of the August 2014 issue of *login*: about the potential effects of Wassenaar.

Both speakers called for creating textbook definitions for words relating to security, such as “exploit” and (better) “weaponized exploit.” Having these terms defined by lawmakers or journalists may result in making much security research illegal in most countries.

Shifting into the final topic, Sergey explained that software insecurity has everything to do with attempting to solve a problem that is unsolvable. The Chomsky-Schützenberger hierarchy (1956) describes sets of parsers that are required to recognize the four classes of formal grammars. Only regular grammars can be proven correct, and the use of any other more complex grammars leads to parsers that cannot be proven correct. Parsers handle input to software, and that's exactly the point where exploitation occurs. Too bad the Wassenaar Arrangement doesn't make context-free, context-sensitive, and unrestricted grammars in parsers illegal, as that would actually do a lot to improve software security.

Sergey stated that they wished to patch the Postel Principle: Instead of being liberal in what is accepted in input, input must be matched exactly. FX commented that we needed Postel's Principle to get TCP/IP off the ground. But now we must play by the rules of adults. They both went on to list a series of recent exploits, all of which are triggered by the parsing of input, including OpenBSD's IPv4 bug in 2007, the chunk encoding bug that appeared first in Apache in 2003, then again in Nginx in 2013, goto fail in Mac OS, and Heartbleed in 2014.

They ended by suggesting that people check out the papers from the LangSec workshop at IEEE Security and Privacy 2014: <http://spw14.langsec.org/>.

Steve Bellovin (Columbia) agreed with the point about the need for input recognizers. He provided the example of a fascinating bug in the FTP recognizer, which used a YACC grammar to parse input and led directly to the security hole. Steve concluded that the wrong type of grammar was used for the task. Sergey agreed with Steve's point, and commented that we focus mostly on computation at the syntactic side, but not on the semantic side. FX joked that if they hadn't used YACC, but the typical approach of using case statements, they would have had hundreds of bugs.

Attacks and Transparency

Summarized by Brendan Saltaformaggio (bdsaltaformaggio@gmail.com)

On the Practical Exploitability of Dual EC in TLS Implementations

Stephen Checkoway, Johns Hopkins University; Matthew Fredrikson, University of Wisconsin—Madison; Ruben Niederhagen, Technische Universiteit Eindhoven; Adam Everspaugh, University of Wisconsin—Madison; Matthew Green, Johns Hopkins University; Tanja Lange, Technische Universiteit Eindhoven; Thomas Ristenpart, University of Wisconsin—Madison; Daniel J. Bernstein, Technische Universiteit Eindhoven and University of Illinois at Chicago; Jake Maskiewicz and Hovav Shacham, University of California, San Diego

Stephen Checkoway from Johns Hopkins University began this presentation with a central assumption: An attacker can generate the constants used in Dual EC. Based on this assumption, the research aimed to analyze the cost of such attacks against TLS implementations (such as that used in SSL libraries) that use NIST's Dual EC pseudo-random number generator. After reminding the audience that the NSA has gone to great lengths to standardize Dual EC implementations, Checkoway shifted to a brief overview of the operations of Dual EC. Checkoway used diagrams to show where the product of each iteration becomes the seed of the next pseudo-random number generation, and by leaking that seed, it's easy to derive the next output of Dual EC.

In this work, the authors analyzed four common TLS libraries: RSA BSAFE for Java and C/C++, Microsoft Secure Channel, and OpenSSL-FIPS. Checkoway detailed how their attack, assuming the back door introduced by knowing the Dual EC constants exists, could be implemented. Next, the effectiveness of this attack against each of the four common TLS libraries was shown. Finally, a live demo was shown where data taken from a TLS connection via tcpdump was decrypted in real time in just 3.5 seconds.

The presentation drew a question from Jeremy Epstein, who made it clear that he was from NSF and not NSA. He asked whether this work had been shown to Dickie George, an ex-NSA employee who said that breaking Dual EC was impossible. Checkoway answered that the work had not but that hopefully this dialog could occur in the future.

iSeeYou: Disabling the MacBook Webcam Indicator LED

Matthew Broucker and Stephen Checkoway, Johns Hopkins University

Matthew Broucker reminded the audience that, increasingly, everything around us (from cars to toasters) is being fitted with processors. The MacBook's iSight webcam is no exception. After obtaining an iSight camera from a 2008 MacBook, the research aimed to answer two questions: (1) Can the iSight's firmware disable the LED? and (2) Can the host system replace the iSight's firmware with malicious firmware?

Unfortunately, both challenges are possible. By manipulating the RESET register on the iSight, malicious firmware can disable the LED light regardless of the camera's state (on or standby). Even more disturbing is that the iSight's firmware can be replaced by a non-root process running on the host. Using these

two vulnerabilities, the presentation concluded with a demo of videoing the audience with an iSight camera and proof-of-concept application. During video display, the speaker enabled and disabled the LED light.

The questions mainly focused on solutions for the attack. Andrew West (Verisign Labs) asked whether a fix other than mechanically binding the LED to the camera's operation exists (such as, a hybrid hardware-software solution). Broker's reply was very practical: While many solutions exist, the devices are already widely used and thus it will be difficult to push a solution out to so many vulnerable devices. Someone asked whether they had informed Apple of their results. Broker answered yes, but that the devices are already deployed. He also noted that the camera used in this work was from 2008 and that new cameras are different but may still be exploitable. Jeremy Epstein pointed out that the Sun4 workstation in 1994 had the same problem.

From the Aether to the Ethernet—Attacking the Internet Using Broadcast Digital Television

Yossef Oren and Angelos D. Keromytis, Columbia University

Yossef Oren gave a highly entertaining presentation of how HbbTVs suffer from a trio of security vulnerabilities. First, users have no control over the life cycle of applications running on the TV. Secondly, Web-origin policies are specified by the application itself. Finally, the RF-distributed code is allowed access to Internet resources. The presentation focused on one of multiple attacks presented in the paper: injecting attack code into a TV radio signal.

In this attack, Oren described how an attacker could intercept a standard TV radio signal. Attack code could be injected into this signal and then rebroadcast from an attacker-controlled antenna. Any HbbTVs picking up this signal could then be used to execute the attacker's code and access the Internet to download more code or to perform various wide-scale attacks.

Oren concluded with a number of countermeasures for each of the three HbbTV security problems. However, Oren also noted that enacting such countermeasures would incur cost to the HbbTV provider—making them unlikely to be adopted. One questioner asked whether regulators were concerned about such attacks. Oren responded that previous research had looked into the privacy implications of HbbTV and that he was aware of some concern within the European Union. Another wondered whether user credentials could be stolen in these attacks. Oren said if they had been previously cached, they could be stolen.

Security Analysis of a Full-Body Scanner

Keaton Mowery, University of California, San Diego; Eric Wustrow, University of Michigan; Tom Wypych, Corey Singleton, Chris Comfort, and Eric Rescorla, University of California, San Diego; Stephen Checkoway, Johns Hopkins University; J. Alex Halderman, University of Michigan; Hovav Shacham, University of California, San Diego

Keaton Mowery opened this talk with some background of how his lab obtained a Rapiscan Secure 1000 Full-Body Scanner—naturally, from eBay. Upon arrival, the researchers aimed to

answer three questions: (1) Is the Secure 1000 radiologically safe? (2) What privacy safeguards exist? and (3) How effective is the Secure 1000 at detecting contraband? After a brief explanation of x-ray physics, Mowery explained how the Secure 1000's imaging equipment operates, and what results they obtained. The physics is important, because backscatter radiation is only created by less dense matter, such as flesh, as opposed to more dense material, like the metal in guns or knives, which does not produce backscatter, and appears black on scans.

On the questions of radiation safety, they found that the Secure 1000 emits safe levels of radiation. Thanks to a simple, modular software design, an attacker would need physical access to replace the system's ROM to over-irradiate a scan subject. The remainder of the talk focused on the Secure 1000's contraband detection capabilities (or lack thereof). Several slides presented side-by-side images of a scan subject concealing handguns and plastic explosives, all completely indistinguishable from unarmed scan subjects. The talk concluded by asking for more open evaluation of the full-body scanners in use by the TSA. Mowery suggested visiting <https://radsec.org> for more information.

Andrew Drew (Qualcomm) asked what tradeoff the TSA may be making to deploy working systems more quickly. Mowery acknowledged that the devices were not perfect but perhaps better than nothing. One questioner asked whether image processing may be able to detect hidden contraband that the human eye cannot. Mowery replied that this was not tested and that he doubted it would help. Later, Mowery was asked how physically obvious the hidden contraband was (since the presentation only showed x-ray images). He responded that it depends on how small the contraband is: A knife was possible to hide, but hiding a handgun was difficult to do inconspicuously.

ROP: Return of the %edi

Summarized by Ben Stock (ben.stock@fau.de)

ROP Is Still Dangerous: Breaking Modern Defenses

Nicholas Carlini and David Wagner, University of California, Berkeley

Nicholas Carlini pointed out that even enhanced versions of these countermeasures can be bypassed using only gadgets in simple coreutil tools like diff. He discussed the fact that all approaches that rely on lightweight Control-Flow Integrity suffer from the aforementioned issues and new defenses need to be proposed.

In the Q&A, an attendee asked whether their outlined attacks work on both x86 and x64. Carlini replied that the exploits discussed in their paper were targeting both these architectures and that there are not fundamental differences between the two.

Stitching the Gadgets: On the Ineffectiveness of Coarse-Grained Control-Flow Integrity Protection

Lucas Davi and Ahmad-Reza Sadeghi, Intel CRI-SC at Technische Universität Darmstadt; Daniel Lehmann, Technische Universität Darmstadt; Fabian Monroe, The University of North Carolina at Chapel Hill

Lucas Davi showed that even by combining the most strict rules that have been proposed over the last few years to a so-called ÜberCFI, kernel32.dll still carries enough long, call-preceded gadgets to be Turing-complete.

A question that arose was how hard the long NOPs were to find. Davi pointed out that while not all sequences that might be used are without any side effects, many other gadgets exist that reverse the side effects. Therefore, combining two gadgets of such characteristics again leads to a NOP gadget. Another question concerned the type of initial exploit step used by the exploits presented by Davi, to which he replied that they relied both on stack as well as heap overflows.

Size Does Matter: Why Using Gadget-Chain Length to Prevent Code-Reuse Attacks Is Hard

Enes Göktaş, Vrije Universiteit Amsterdam; Elias Athanasopoulos, FORTH-ICS; Michalis Polychronakis, Columbia University; Herbert Bos, Vrije Universiteit Amsterdam; Georgios Portokalidis, Stevens Institute of Technology

Enes Göktaş focused on breaking existing countermeasures but also on analyzing parameters for the existing approaches with which they would work. In doing so, their work found that it is possible to mitigate the effects of vulnerabilities by tuning the parameters for kBouncer (maximum length for a sequence to be seen as a gadget and number of gadgets used) for specific applications. Nevertheless, no generic parameters could be found that would neither cause false positives nor effectively stop the attacks. They pointed out that applications need to be analyzed in advance to determine parameters for the protection schemes to properly work.

Oxymoron: Making Fine-Grained Memory Randomization Practical by Allowing Code Sharing

Michael Backes, Saarland University and Max Planck Institute for Software Systems (MPI-SWS); Stefan Nürnberger, Saarland University

Stefan Nürnberger noted that although ASLR is a viable technique to make an attacker's life harder due to unguessable addresses, it can be bypassed if just one address from a library is somehow leaked (since the libraries are in memory en bloc). One approach to counter this is to spread out libraries across memory. This, however, impairs the sharing of a library between processes since relative calls to other library functions must be replaced with absolute ones (as there is no longer a correlation between the addresses of different functions). On a recent version of Ubuntu, this effectively leads to 1.4 GB being wasted for duplicate libraries when the OS is fully loaded.

Nürnberger proposes to solve this by using a combination of segmentation and lookup tables for all functions. Inside a library, a function can then place an indirect call to the n -th function in a table. The address of that table does not have to be known

during compile time, but rather is set in the segment register at runtime. In doing so and in putting the lookup table in a memory region that is not normally accessible by code, the proposed solution provides randomization of the address space as well and allows for code sharing at the same time. In total, their approach has a runtime overhead of up to 3.5% as well as 13.5% file size overhead.

Nürnberg was asked if the approach would also work for other architectures, such as ARM. To his mind, the system would work but would require more instructions (as ARM is a RISC architecture). One potential attack was brought up by another questioner, namely scraping pointers to said library functions from memory. In this case, since the addresses need to be on the stack for the matching returns, the addresses of single library functions could be leaked. Someone asked about implementation of the mechanism. Nürnberg outlined that the approach can be easily built into a generic compiler like gcc. The final question concerned its feasibility on x64 systems. Nürnberg replied that while there are differences related to segmentation between x64 and x86, the approach would work as well; the only pitfall was the fact that the address of the lookup table could no longer be hidden (due to the differences in segmentation).

Safer Sign-Ons

Summarized by Venkatanathan Varadarajan (venkatv@cs.wics.edu)

Password Managers: Attacks and Defenses

David Silver, Suman Jana, and Dan Boneh, Stanford University; Eric Chen and Collin Jackson, Carnegie Mellon University

David Silver presented various vulnerabilities in many commercially available password managers like LastPass, 1Password, KeePass, etc. that aim to provide user convenience but often end up compromising security in previously unforeseen scenarios. Silver and his team particularly focused on the poorly named password manager feature called *automatic autofill*. This feature proactively fills any previously seen username, password pairs on Web sites without user interaction as opposed to manual autofill, which requires user interaction. It is not always safe to automatically autofill passwords. For example, autofilling passwords on a page that suspiciously fails to provide a valid SSL certificate or when the HTML form action URL changed between the time the password was saved and when it is used may not always be secure. Silver pointed out that some password managers automatically autofill in these scenarios.

Silver detailed one particular attack scenario where a malicious coffee shop owner providing free WiFi service could steal passwords from the password manager for a totally unrelated Internet activity. He showed a prerecorded demo video where an innocuous visit to an online pizza ordering service could seamlessly redirect to a totally unrelated Web site (e.g., AT&T or an online banking Web site) with malicious JavaScript code embedded. This action could trick the password manager into autofilling the credentials to the embedded malicious code, which could

save the credentials on a remote server. The malicious code redirects the user back to the user-requested Web site, completing all this in milliseconds and remaining visually unobservable to the user. All password managers using the automatic autofill feature were vulnerable to this attack. Silver also discussed various defense mechanisms against these attacks. He and his team observed that disabling automatic autofill (i.e., manual autofill) is immune to these attacks. Particularly, doing manual autofill and submit is both secure and more convenient to users because it requires only one click. A better and more secure alternative to this is what he called *secure filling*; among other constraints JavaScript code was not allowed to read autofilled contents. They implemented a prototype on the Chromium browser that required only 50 lines of code.

Following the talk, David Wagner (UC Berkeley) observed that the secure password filling defense might have usability costs associated with it as there are benign Web sites that want JavaScript code to read autofilled contents. He asked whether the team looked at ways to reduce this cost by remembering whether a JavaScript code tried to read the autofilled contents while saving. Silver agreed with this suggestion, although he pointed out such a mechanism may not be able to always protect users from malicious JavaScript code. Jeffrey Goldberg (AgileBits) posed multiple questions. First, he asked Silver how frequently the action URLs change in reality. Silver pointed out that irrespective of the frequency, it is important to observe that the action URLs should be from the same origin (or domain). Second, Goldberg felt that the secure autofilling feature seemed to be rigorous in not allowing JS code to read autofilled content and asked how many benign Web sites were affected. Silver responded that they found only 10 of the top Alexa Web sites that used AJAX were affected. Thirdly, when asked about the impact of the work on commercial password managers, Silver mentioned a couple: 1Password now warns about autofilling when there is a SSL certificate validation failure, and LastPass stopped autofilling passwords in forms that are displayed in an iFrame.

The Emperor's New Password Manager: Security Analysis of Web-Based Password Managers

Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song, University of California, Berkeley

Zhiwei Li started out by mentioning the importance of password managers and their popularity but questioned their security. Particularly, the authors identified four important security properties any password manager should provide: master account security, credentials database security, collaborator integrity, and unlinkability (a property of a password manager that does not allow tracking of a user across Web sites). Li went on to present four classes of vulnerabilities that their research uncovered that none of the password managers were immune to.

First, Li observed that all password managers provide a bookmarklet feature and these bookmarklets often run in an insecure environment, which may include running in the context of a

malicious JavaScript (JS) code. Second, he presented a vulnerability where a Cross-Site Request Forgery (CSRF) would let an attacker choose an arbitrary one-time password (OTP) which could, in turn, be used to hijack the master account. Third, Li presented an example of another class of attack where sharing an asset/credential between users could result in unintended disclosure of credentials of the user who initiated the collaboration. Li pointed out that such vulnerabilities arise as a result of mistaking authentication for authorization. Fourth, Li showed a pre-recorded video of a phishing attack (under the class of User Interface vulnerabilities) where Li and his team were able to phish for LastPass's master credentials while remaining unnoticeable to the user. Li concluded the talk mentioning that there is no single solution that could solve all these damaging vulnerabilities, and it might take years for password managers to mature.

Jeffrey Goldberg (AgileBits) asked how the commercial password manager responded to their vulnerabilities. Li responded that some of them patched their code to defend against certain vulnerabilities but not for others. Goldberg then asked for Li's opinion on how single sign-on services compare to password managers. Li responded that similar vulnerabilities may still haunt such services, and both require further research to make them secure.

SpanDex: Secure Password Tracking for Android

Landon P. Cox, Peter Gilbert, Geoffrey Lawler, Valentin Pistol, and Ali Razeen, Bi Wu, and Sai Cheemalapati, Duke University

Landon Cox, on behalf of his students, presented their research project SpanDex, a tool that uses taint-tracking of passwords to hunt down inappropriate use of passwords and phishing attempts in mobile apps. Cox first showed various examples of phishing apps that steal user credentials used for the real counterparts, for example, the Wroba Android app that steals banking credentials.

Cox provided a simple introduction to taint-tracking where a variable/data-item that one wishes to track is tagged and followed using data-dependency and taint-transferring or propagation logic. Particularly, there are two flows in the taint-propagation logic: explicit flows that transfer the taint because of direct assignments operations, and implicit flows that transfer taint because of control flow or complex interactions between variables. Cox and his team identified the latter as significantly harder to track but essential to get good coverage and security guarantees. One of the challenges in taint-tracking in these implicit flows is that it often results in over-tainting. Cox pointed out various ways to avoid over-tainting implicit flows by weighting the taints differently. He used an example to motivate this observation: a tainted variable `s` used in a condition, `s == 0`, does not leak much information compared to an explicit flow tainting. Similar optimizations were used in making the taint-tracking faster and efficient. Cox and his team used a symbolic execution tool to do the taint-tracking and prototyped an implementation of SpanDex for checking Android applications.

Jeffrey Goldberg (AgileBits) asked what kind of passwords they look at in this work and did all of them follow power law. Landon Cox responded that all passwords were strings of characters and nothing else. Someone asked whether they looked at passwords that were processed locally since all the examples that were mentioned involved sending password on the wire. Cox replied that among 50 applications that they looked into none did local processing of passwords. Finally, someone asked why Cox and his team restricted their evaluation to uniform or Zipf distribution of passwords and not a frequency-based distribution. Cox responded by recalling that the password data set they used in the evaluation consisted of unique passwords, and a frequency-based distribution is not possible using that data set.

SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities

Yuchen Zhou and David Evans, University of Virginia

Yuchen Zhou first introduced the concept of single sign-on where there is an identity provider (like Facebook, Google) who maintains user credentials and a third-party integrator who wants to authenticate a user's identity. Although the SSO SDKs claim that integrators require little or no knowledge of security expertise, Zhou's research shows that this is often not true and might result in various vulnerabilities such as credential misuse. One instance of such a vulnerability is the misuse of an access token provided by the identity provider. A malicious user could reuse an access token provided for a particular application to access a different and completely unrelated application if the application fails to check the application ID provided in the access token. Zhou pointed out various real applications that were vulnerable to this attack and how trivially they expose these access tokens.

Zhou next presented SSOScan, which scans Web sites (integrators) for both credential misuse and credential leakage. SSOScan consists of three components: the Enroller, which automatically registers and logs into the Web page under test; the Oracle, which verifies successful enrollment and confirms session authentication; and the Vulnerability Tester, which tests for vulnerabilities using the newly registered account. Zhou discussed various challenges with automating the Enroller and Oracle. Zhou then presented the evaluation results on the top approximately 20k QuantCast sites: Out of 1.7k sites that use Facebook's SSO, 20.3% of them had at least one vulnerability. Zhou also pointed out that 12.1% misused credentials and 8.6% leaked credentials, with 2.3% of them having buggy implementation. When looking at the correlation between number of vulnerabilities and popularity of the Web site, Zhou found that the number of vulnerabilities found by SSOScan were the same irrespective of the popularity of the Web site. Zhou and his team also contacted many vendors about these vulnerabilities, and some responded with a fix. Zhou concluded by releasing the SSOScan as a service where vendors could check their Web site implementations for various vulnerabilities disclosed in this research (available at www.ssoscan.org).

Someone asked why they only chose 9.3% of 20k sites. Zhou responded that they randomly chose the Web sites and also restricted themselves to US sites since they were constrained by the language used in the sites. Another person inquired whether this service could be misused with malicious intent. Zhou replied that vulnerable Web sites were not publicly disclosed and to the best of their knowledge there seemed to be no such misuse. When asked whether changing APIs help to make misuse or leaking credentials harder for the developers, Zhou responded that developers often fail to read the documentation before implementing and was not sure whether changing APIs would help in such cases. Finally, someone asked whether any of the reported vulnerabilities were exploited in the wild. Zhou did not know of any such exploitation, but he believed that there were highly sensitive sites (e.g., match.com) that were still vulnerable and waiting to be exploited.

Passwords

Summarized by Andrei Costin (costin@eurecom.fr)

A Large-Scale Empirical Analysis of Chinese Web Passwords

Zhigong Li and Weili Han, Fudan University; Wenyuan Xu, Zhejiang University

Zhigong Li began his talk with the observation that when surfing the Web, a user often needs to register an account and that requires a password. Another observation was that password choice has strong geo-location influence. Zhigong attempted to answer the following two questions: Do Chinese Internet users have better passwords than others? How can one efficiently guess their passwords? He also observed that Chinese users form the biggest Internet group, with over 600 million netizens.

Zhigong mentioned they used leaked password databases from the top five Chinese Web sites as well as from Yahoo! and RockYou leaks. Their methodology was based on characters used, patterns, and their analysis. They found that most popular in both Chinese and English leaks were passwords 123456 and 123456789. However, digits are more common in Chinese passwords than in English ones. This is also because in Chinese the pronunciation of some digits is similar to letters. For example, pronouncing the number “520” sounds like “I love you” in Chinese.

Zhigong then presented their analysis on the resistance to guessing Chinese passwords. He explained they used “alpha work factor” analysis, which is the number of guesses required for a given success probability alpha. In many cases the alpha work factor can be very high. They found RockYou and Yahoo! passwords have higher work factor for $\alpha < 2.5$.

Zhigong also explained that Chinese characters are input using pinyins, compared to simple characters in English or other western languages. About 25% of Chinese passwords contain pinyins. Interestingly, the top Chinese pinyin among Chinese passwords is “woaini,” which stands for “I love you.”

Zhigong suggested that dates also play an important role in password formation. In Chinese passwords most dates are formatted as YYYYMMDD, while dates in English passwords appear as MMDDYYYY. An additional finding is that dates in both Chinese and English passwords are put at the end.

David Wagner (University of California, Berkeley) asked whether it's possible to compare probabilistic CFG (P-CFG) to the password cracking tools (simpler to use, etc.). Zhigong answered that cracking tools are dictionary based; P-CFG can analyze the structure of the password and can add some rules (e.g., dates), while cracking tools cannot add rules and hence are harder to use and less effective. Someone asked how dates have been distinguished from other random numbers in the analysis. Zhigong explained that there of course could be false positives. While the approach is simpler for eight-digit dates, for six-digit dates it can be trickier, so some six-digit dates were removed from the training.

Password Portfolios and the Finite-Effort User: Sustainably Managing Large Numbers of Accounts

Dinei Florêncio and Cormac Herley, Microsoft Research; Paul C. van Oorschot, Carleton University

Cormac Herley started at a fast pace presenting the following observation: Although everyone knows the basic rules, that passwords should be random and should not be reused across accounts, virtually nobody follows them.

He then presented a simple calculation showing that to remember 100 different five-character passwords for 100 different accounts would require the user to remember more than 4000 bits!

To reduce this burden, the user could weaken the password, which means reducing the $\lg(S)$. However, the computations show the amount of information is still way to high to remember (i.e., 524 bits), even if $\lg(S)$ is zero! The hypothesis Cormac advanced is that group and reuse are the only way out for normal people. He then mentioned that there were many ways to organize a password portfolio. For example, doubling the number of passwords more than halves the password strength, while stronger passwords force more password reuse.

Cormac emphasized that the question is not “are longer passwords stronger than shorter passwords?” nor “how does one generate secure random passwords or mnemonics?” From their research perspective, the question is “how does one minimize the password portfolio's expected loss?” For example, setting the effort to infinity minimizes the probability of harm in case of loss of password. Cormac, however, highlights the fact that users also care about the effort. So the true question is “how does one minimize the portfolio's expected loss + the user effort involved?”

Cormac explained that such a question makes the user think the right thing to do is to have some accounts that are weakly protected. For example, that “123456” is a top password in the RockYou database reveals that maybe those users chose to spend the password effort on something that matters more. However,

passwords unfortunately reveal many other things, which means the risks are not independent across the accounts and not dependent only on the strength. In addition, the risk to the i -th account also depends on external factors such as malware and keyloggers.

Cormac provided the criteria for optimality (for loss, effort, and password policies). First is the division of things into G different groups. At this stage, how to optimally divide things into groups and how to choose division boundaries between groups are open questions. Second is that groups adjacent to each other should have similar weighted loss: for example, you shouldn't "put all your most important stuff into the same basket."

Cormac concluded that random and unique passwords are infeasible for large portfolios and that the user's interest is to minimize $L(\text{oss})+E(\text{ffort})$ rather than just $L(\text{oss})$ over a portfolio. He also concluded that strategies that exclude reuse or exclude weak passwords are both suboptimal. The final conclusion from Cormac is that the best strategy in password grouping is: high-value accounts with strong passwords (low probability of compromise) and low-value accounts with lower password policies (high probability of compromise).

Cormac was asked about the needed effort to remember which password is in which accounts' group out of G groups. Cormac confirmed that such effort is required and referenced a formula in the white paper. In response to another question, Cormac pointed out that this model is a simplification and that there are many other factors that might need to be considered. For example, how to choose optimal G is not part of this research.

Telepathwords: Preventing Weak Passwords by Reading Users' Minds

Saranga Komanduri, Richard Shay, and Lorrie Faith Cranor, Carnegie Mellon University; Cormac Herley and Stuart Schechter, Microsoft Research

Saranga Komanduri introduced the authentication eco-system, which, in his model, comprises users (creating passwords), attackers (hijacking users' passwords), and admins (blocking attackers and protecting users). He also noted that admins create password policies to protect users. It is long known that simple password policies do not solve the issue. For example, `Qwerty123456` or `Thisismypassword!` adhere to policy but still can be viewed as weak. He also noted that in systems running Microsoft's AD, the three-class policies did not seem to have a notable effect in increasing the security of those systems.

Saranga presented his team goal as focusing on the weakest passwords. He also mentioned their contributions. One, they have shown that character requirements do not prevent weak passwords. Another, they detect weak passwords with guessability and real-time feedback.

Saranga went on to present their system, Telepathwords. It is similar to auto-complete, but its goal is to prevent the input of weak passwords. He underlined the fact that showing the user that a machine can guess the passwords is a good demonstration that attackers could do the same. He pointed out that

password policies have not changed much since 1979, when the six-character password policy was proposed. At the same time, password strength meters are typically based on character requirements, and there is no consistency across different meters. Even though the "zxcvbn" open source meter, which estimates the entropy of passwords, was introduced to improve all the above, these meters still don't explain their scores.

Saranga explained they generated predictions using multiple models, including N -gram models, keyboard layout modes (e.g., similar to what some password cracking software does), repetitive passwords (e.g., `abcabc`), and interleaved strings (e.g., `p*a*s*s*`). The system can also be extended with more and better predictors. Then they created multiple policy conditions for a randomized controlled study. The study was designed as a hypothetical email scenario for password creation.

Saranga presented several policy metrics they used to assess the results. They used Weir+ guessability (i.e., finding single weakest password in the sample), zxcvbn entropy estimation (i.e., minimizing entropy in each sample set), probability conditions, usability metrics for creation of the password, and recalling of the password after several days of experiment.

Finally, Saranga presented some of the results. For example, dictionary-based policies are much better than their simple character counterparts. A real surprise is that `3class8` is not essentially stronger than `basic8` with regards to guessability. Also, the policy did not affect the recall metric after 2–5 days. As for the creation time metric, Telepathwords policies took the longest time (90 seconds) compared to an average of 20–40 seconds for other policies.

Saranga concluded that character class policies had little to no effect in creating stronger passwords (i.e., less guessable ones). Additionally, dictionary policies' real-time feedback can help users create stronger passwords, but incurs a usability cost at creation time. Telepathwords was found to help users understand why their passwords were weak.

In response to a question, Saranga confirmed that Telepathwords are not harder to remember compared to random password generators according to their results, although they take a little bit longer to create and read the system's feedback. David Wagner (University of California, Berkeley) asked whether it's possible to download Telepathwords code and use on Web sites. Saranga said that Telepathwords is up and online, and that anyone can use it.

Towards Reliable Storage of 56-bit Secrets in Human Memory

Joseph Bonneau, Princeton University; Stuart Schechter, Microsoft Research

Stuart Schechter started with the observation that a user-chosen secret can never be provably hard to guess. Hence such a provably hard-to-guess secret has to be randomly generated by the system, for example, by a 56-bit secret key. There are multiple scenarios when a 56-bit strong key is required. One example is

the master password for password managers. Another example is the access to organizations with a large number of users where, for example, weaker passwords cannot be filtered out and give attackers easier access to the organization.

Stuart pointed out that such an approach does not mandate that all Web sites now start using such unique 56-bit secrets, but that they be used only for critically important cases. Stuart suggested that we all use metaphors to explain problems, but the fact is that writing to the brain is harder than writing to the hard disk, and these metaphors can sometimes obscure reality.

Stuart then noted that our brains are designed to forget random data seen only once. He also noted that we have all learned through spaced repetition. We use repetition to learn important things, and we know it works great for learning. So the question Stuart and his team tried to address is whether humans could apply the same learning through spaced repetition to a 56-bit secret and how to apply this to remembering the secrets.

Stuart explained the setup of their experiment that tried to answer this question. The experiment involved subjects recruited via Mechanical Turk. The learning and recalling was camouflaged as a login to the system, which was presented to subjects as an experiment for something else (to avoid subject bias, suspicions, tricking). The 56-bit secret was presented for learning and recalling in three groups of four characters each. Some subjects had groups formed of random letters, while other subjects had groups formed of meaningful words. The system was designed close to reality, requiring the subjects to log in around 10 times a day, hence simulating an average workday in an enterprise.

Stuart then presented the results. Four subjects stopped learning codes for various and even funny reasons. On average, the subjects learned the entire secret at their 14th login attempt. For all subjects, learning the first group/code took most of the time. This could be for multiple reasons, such as not being used to the system or not understanding the system well enough at the beginning. Stuart presented the result that in this system there was only a 12% password forget rate registered compared to 26% in Telepathwords. Another finding was that recall rates decreased after more than two weeks. Finally, the recall rate for passwords grouped in word groups was 62% compared to 56% of the passwords grouped in random letters groups.

Andrei Costin (Eurecom) asked whether group interference across multiple 56-bit secrets (i.e., for multiple secrets, which group goes where and to which secret) was studied and how subjects responded to these interferences. Stuart explained that definitely interference is a potential direction for study, but they did not study this at present. He added that in this single 56-bit secret experiment, the results show that 2nd and 3rd group (letters, words) did not interfere inside this single 56-bit secret. He finally added that interference should not be a problem, since it is expected that a normal user should require only two or a maximum of three such 56-bit secrets during their lifetime.

Web Security: The Browser Strikes Back

Summarized by Alexandros Kapravelos (kapravel@cs.ucsb.edu)

Automatically Detecting Vulnerable Websites Before They Turn Malicious

Kyle Soska and Nicolas Christin, Carnegie Mellon University

Awarded Best Student Paper!

Kyle Soska presented a novel system that aims to predict which Web sites will become malicious in the future. There are many challenges in achieving such a difficult task. The target data set is the entire Web, a continuously growing data set of billions of Web pages. Moreover the data set is highly unbalanced, with many benign Web pages and a few malicious ones, the labels of the data set are incomplete, and there is no ground truth. Additionally, just predicting a Web site as potentially malicious in the future is not so useful on its own; webmasters need to be aware of the reasoning behind such a prediction so that they can react preventively. Lastly, the Web evolves over time; attacks change as new vulnerabilities are discovered, and the system should be able to react and adapt to these changes.

To cope with these challenges, the authors created a classifier based on C4.5 decision trees. With the use of blacklists and archive.org they created a data set of soon-to-be malicious Web sites and benign Web sites. The authors managed to isolate user-generated content from the visited pages with the use of composite importance, so that the system is able to focus only on the template-generated part of the Web pages, where vulnerabilities might exist. With a combination of dynamic and static features the system is able to achieve up to a 66% true positive rate with 17% false positives, showing this way that predicting malicious Web pages is possible.

Hulk: Eliciting Malicious Behavior in Browser Extensions

Alexandros Kapravelos, University of California, Santa Barbara; Chris Grier, University of California, Berkeley, and International Computer Science Institute; Neha Chachra, University of California, San Diego; Christopher Kruegel and Giovanni Vigna, University of California, Santa Barbara; Vern Paxson, University of California, Berkeley, and International Computer Science Institute

Alexandros Kapravelos focused on browser extensions: small HTML and JavaScript programs that modify and enhance the functionality of the browser. Alexandros showed that to compromise the browser, the attackers no longer need a 0-day exploit, but they can gain sufficient access in the user's system through a malicious extension. These extensions can inject more advertisements or perform affiliate fraud among other things.

To deal with this problem the authors developed Hulk, a system that dynamically analyzes and automatically detects malicious browser extensions. They introduced the notion of HoneyPages, which are dynamic pages that change on the fly to match what the currently analyzed extension is looking for in the content of a visited page. Hulk identifies malicious behavior by monitoring all aspects of the extension's execution. For example, Hulk will detect if the extension is preventing the user from uninstalling

it or if it is stealing login credentials from forms. Hulk has analyzed 48k extensions and found 130 to be malicious and 4,712 of them suspicious. In the last part of the talk, Alexandros elaborated on the needed extension architecture changes that could either ease the analysis of extensions or prevent certain types of malicious extensions.

Precise Client-Side Protection against DOM-based Cross-Site Scripting

Ben Stock, University of Erlangen-Nuremberg; Sebastian Lekies, Tobias Mueller, Patrick Spiegel, and Martin Johns, SAP AG

Ben Stock focused on current defenses against DOM-based XSS attacks and how those can be circumvented automatically. By going over an extensive list of the current limitations of string-based XSS filters, the authors showed that the current state-of-the-art in DOM-based XSS can be evaded. Moreover, they implemented an engine that will automatically exploit 1,169 out of 1,602 real-world vulnerabilities despite the current defenses.

Ben made the simple, yet powerful, observation that client-side XSS filters use string comparison to approximate data flow, but this is unnecessary since it happens on the client side. Therefore, they propose a taint-enhanced JavaScript engine that tracks the flow of attacker-controlled data in a combination of taint-aware JavaScript and HTML parsers capable of detecting generation of code from tainted values. The new proposed method catches every single exploit, yielding no false negatives, and a 0.16% false positive rate for all analyzed documents. Moreover, the performance penalty introduced by the new defense mechanism was between 7–17%, with some optimizations applicable.

On the Effective Prevention of TLS Man-in-the-Middle Attacks in Web Applications

Nikolaos Karapanos and Srdjan Capkun, ETH Zürich

Nikos Karapanos focused on TLS man-in-the-middle attacks in Web applications. In the current state, server authentication can be circumvented by compromising a certificate authority, compromising the server's key (e.g., via the Heartbleed bug) or simply by letting the user click through the certificate warning on her own. Nikos elaborated on how TLS Channel IDs work, which is the current state-of-the-art for defending against MITM attacks. He then showed how one can circumvent TLS Channel IDs by proposing a new attack called man-in-the-middle-script-in-the-browser. This attack works not by impersonating the user to the server, but by injecting back to the user JavaScript code that will run in the context of the user's browser. Any communication with the server from the browser is now properly authenticated since it comes directly from the user's browser, but the code is controlled by the attacker.

To cope with this new type of attack, the authors propose Server Invariance with Strong Client Authentication (SISCA). This novel approach is based on the observation that the client needs to communicate with multiple entities for the MITM-SITB attack to work. By combining SISCA and TLS Channel IDs, Nikos showed that MITM attacks can be successfully prevented.

Poster Session

The summary of this session is available online as an electronic supplement: www.usenix.org/login/dec14.

Side Channels

Summarized by Qi Alfred Chen (alfchen@umich.edu)

Scheduler-Based Defenses against Cross-VM Side-Channels

Venkatanathan Varadarajan, Thomas Ristenpart, and Michael Swift, University of Wisconsin—Madison

Venkatanathan Varadarajan first introduced multi-tenancy in public clouds, which can benefit the utilization and reduce service cost but results in cross-VM attacks due to the difficulty of isolating resources. In their work, the authors targeted Prime+Probe cross-VM side channels that exploit per-core resource sharing: for example, the attack proposed by Zhang et al. in CCS '12, which demonstrated the possibility of extracting ElGamal secret keys. They found that for these attacks to succeed, quick preemption is required on the victim VM, which may be defended by limiting the frequency of VM interactions. This defense idea, which they called soft isolation, allows sharing with low overhead, and at the same time only needs simple changes to the hypervisor's CPU scheduler. After over-viewing the high-level idea, Varadarajan introduced some background about the requirements along with the corresponding reasons for the availability of quick preemption in the cache-based side-channel attacks.

To achieve soft isolation, they proposed using the Minimum RunTime (MRT) guarantee, which is available in Xen and KVM. Varadarajan first presented the security evaluation of the MRT mechanism against cross-VM side channels. In a public cloud-like setting, using victims based on a simple model, they showed that under 1 ms MRT the side channel was not observable by the best known attacker. For ElGamal victims, the number of iterations per preemption was shown to increase dramatically to 32 under 0.5 ms MRT and to 68 under 1 ms MRT. This made those cache-based attacks very unlikely to succeed since they require multiple preemptions within one iteration for noise-reduction.

After demonstrating that soft isolation has the potential to defend against cache-based side-channel attacks, Varadarajan showed the performance overhead for normal applications when using MRT. Under 5 ms MRT, both interactive and batch workloads in the experiments had less than 7% overhead. Varadarajan also claimed that with 5 ms MRT and selective state cleansing (detailed in the paper), the overhead was negligible and no known attacks could work.

Varadarajan was asked whether this defense could be applied to other VM hypervisors besides Xen, and he replied that the minimum runtime support is not specific to Xen; for example, it also exists in Linux. He added that it is not particularly tied to a specific VM hypervisor, although there are still things to work out on other systems: for example, requiring performance

analysis on other systems to achieve low overhead. Varadarajan was also asked how their defense compares to existing defenses, and he answered that previous defense methods mostly require dedicated hardware or else they sacrifice significant workload performance, while their defense is easier to deploy and also has low overhead under the current hypervisor settings.

Preventing Cryptographic Key Leakage in Cloud Virtual Machines

Erman Pattuk, Murat Kantarcioglu, Zhiqiang Lin, and Huseyin Ulusoy, The University of Texas at Dallas

Murat Kantarcioglu motivated their work by various security threats to the cloud VMs because of physical machine resource sharing: for example, many side-channel attacks have the potential of extracting cryptographic keys. To protect the secret key, their work proposes an idea of partitioning the keys into many shares using secret sharing and threshold cryptography, thus making it harder for attackers to capture the complete cryptographic keys. Following the motivation and overview, Kantarcioglu provided some background on secret sharing and threshold cryptography (e.g., Distributed-RSA, Threshold-RSA, and Shamir secret sharing).

Based on the idea of distributing the keys in many pieces, they proposed a system, called Hermes, to prevent the secret key leakage in the public cloud. As a proof-of-concept, they applied Hermes to enhance the protection of SSL/TLS cryptographic keys. In the initialization phase of this prototype, the cryptographic key is partitioned and distributed to a set of defender VMs, and Hermes is then bootstrapped with established initial authenticated and secure SSL channels between pairs of defender VMs. After that, client connection requests are sent to one of the defender VMs, named combiner VM, and the combiner VM will work with other VMs to provide services such as distributed signing and decryption. The owner of the secret key will also periodically create new shares for the same secret keys and re-share them to the VMs. These new shares are independent from the previous ones, and this re-sharing adds more difficulty for attackers to extract the cryptographic keys.

Next Kantarcioglu showed the evaluation for Hermes. Hermes was implemented as a shared library in OpenSSL, and in the experiments Hermes was used in various applications with 10 VMs in Amazon EC2. The results showed that inter-VM communication dominated the overhead, and adding defender VMs could lower the overhead. On a mail server, the overhead was at most 8% when the number of clients varied from 1 to 1000. Following the evaluation, Kantarcioglu also talked about how to formalize a multi-objective optimization problem to better choose the Hermes parameters.

Kantarcioglu was asked about the system performance compared to regular SSL, and he replied that using Hermes did introduce more overhead: for example, for one client the connection time was increased from around 2 ms to around 10 ms. He added that for bigger applications this overhead would be smaller.

Kantarcioglu was then asked about their thoughts on protecting the key-sharing process. He answered that they assume that this is a secret sharing process without any adversarial attacks.

FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack

Yuval Yarom and Katrina Falkner, The University of Adelaide

Yuval Yarom presented a new side-channel attack against the L3 cache, called Flush+Reload, which may exist between processes in the same OS and between VMs in the cloud. Yarom started by introducing the memory-sharing mechanism, which is a popular technique for reducing the overall memory footprint and is considered safe. Yarom also provided background on the cache mechanism, especially the L3 cache, which is shared among processors, and the cache flushing functionality for maintaining cache consistency.

Yarom then talked about their Flush+Reload technique, which exploits cache behavior to infer information on victim accesses to the shared memory. In this technique, the attacker first flushes the memory line, and reloads the line after waiting for a while. The attacker then can conclude the victim memory access behavior: If the reload is short, then the victim does access the memory line during the waiting time; otherwise the victim does not access the memory line.

With the Flush+Reload technique, Yarom then showed how they attacked the GnuPG implementation of RSA. They targeted the memory lines mapped with specific code segments in the RSA program, and thus were able to trace the detailed execution of the victim program. Since the clear bits and set bits trigger different code segments in the program, the attacker can extract the secret key. Yarom showed how they traced the detailed program executions, and also showed the bit errors in their experiments on both the same-OS scenario and the cross-VM scenario. Yarom concluded with potential attack applications such as the default OpenSSL implementations of ECDSA and keystroke timing.

Someone asked how long it takes to successfully extract the key. Yarom replied under a few milliseconds. He was asked twice about whether the scheduler-based defense can defend against this attack, and his answer was no since the scheduler-based defense focuses on per-core sharing-based attacks. He was also asked about how to achieve frequent Flush+Reload. He answered that with the attacker and the victim pinned to different cores, the frequency can be high enough. He was then asked whether they can know which bits are missing in the bit errors, and why KVM missed 30 bits. Yarom replied that they can know the missing bit positions, and the high bit error rate for KVM was due to both the more advanced optimizations of the Xeon processors and the aggressive deduplication used. The last question was about whether the distributed key idea can prevent the attack. Yarom answered that having multiple collocated VMs sharing the key may be a protection for Flush+Reload attack.

Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks

Christopher Meyer, Juraj Somorovsky, Eugen Weiss, and Jörg Schwenk, Ruhr-University Bochum; Sebastian Schinzel, Münster University of Applied Sciences; Erik Tews, Technische Universität Darmstadt

Christopher Meyer started the talk by describing the importance of SSL/TLS in security and privacy, and he stated that according to their work, oracle attacks have returned again in SSL/TLS. He then provided background about the famous Bleichenbacher attack and the handshake protocol in SSL/TLS. The Bleichenbacher attack uses the error messages of the PreMasterSecret (PMS) structure-checking algorithm as an oracle to learn whether the decrypted message of the ciphertext from the attacker started with certain bytes. After that, the attacker can leverage the RSA homomorphic property to adjust the payload iteratively and finally restore the PMS, which can be used to derive SSL/TLS session keys. This problem was addressed in the TLS 1.0 standard by using a new random PMS if there is anything wrong.

Meyer went on to report four new Bleichenbacher side channels discovered by their analysis on several widely used SSL/TLS implementations using their T.I.M.E. framework. The first side channel they discovered was due to an implementation bug in JSSE. They found that inserting 0x00 bytes at specific padding positions would generate a different error message, which can be used as the oracle. They evaluated the attack and found that 2048 bit keys were cracked using about 177k queries over a 12-hour period.

While the first side-channel exploits different error messages, the other three new side channels use different processing time as the oracle. The second side channel is likely to exist due to the countermeasure for the original Bleichenbacher attack in the OpenSSL implementation. This is because the new random PMS is generated only when there is something wrong, leading to the timing difference which leaks information about the SSL/TLS compliance of the received PMS. However, they could not execute a practical Bleichenbacher attack, because of the weakness of this oracle. The third side channel is related to the internal exception handling in JSSE. In the implementation, if the message format is not correct, an additional exception will be provoked. Although this does not trigger error messages, it can create timing difference due to the exception handling delays in Java. In the evaluation, this attack took around 19.5 hours and 18.6k queries.

The last side channel also exploits timing differences, but it exists due to hardware issues. They found that the F5 BIG-IP and IBM Datapower, which both use the Cavium NITROX SSL accelerator chip, do not verify the first byte of the message, and additionally found timing differences in processing TLS requests. Meyer showed the evaluation results, and for 2048-bit key this side channel attack took 40 hours and 7371 queries to succeed. In the end, Meyer used a table to summarize the four newly discovered side channels and their attack efficiency.

Meyer was asked whether their toolkit is made publicly available. He replied that their tool has not been released yet. He added that the timing measurement unit of the framework cannot be used ad hoc, because it has to be adjusted and tweaked for each target and environment.

Invited Talk

Summarized by Janos Szurdi (jszurdi@andrew.cmu.edu)

Battling Human Trafficking with Big Data

Rolando R. Lopez, Orphan Secure

Lopez began by discussing the models of human trafficking organizations in different countries. Lopez was an FBI agent for 15 years. During his years at FBI he gained experience about money laundering, drug trafficking, police corruption, and human trafficking. His organization focuses on four methods to battle human trafficking: Prevention and Awareness, Identification and Intervention, Restoration Care, and finally Policy and Law.

The root of human trafficking in many countries is poverty. The Chinese human trafficking model is one such example. From small and poor villages, young women and men are smuggled to big cities all around the world, with the consent of the family and with the promise of a better life. For women, most often work is offered in massage parlors, but when these women get there, their “employees” take all of their documents and they are forced into prostitution. The Chinese mafia runs underground banking and uses wire transfers, making it really hard to cut their money supplies.

In post-Soviet countries, like Russia and Ukraine, human trafficking is very brutal, where children are often sold from orphanages directly into the sex trade. Restoration of victims is really hard in these countries because these victims have no families. As opposed to the Chinese models where the human traffickers try not to harm the girls since they need to maintain a good connection with the villages, in these post-Soviet countries there are no such constraints on the violence. The only chance to help these children and women, who became victims, is for local communities to help them to regain a normal life.

In Thailand the basis of the human trafficking model is the terrible poverty, where families go hungry in certain regions because the crop wasn't enough for the entire year. This is the time when human traffickers go to these villages and offer money to the head of a family for their children, and when the fathers have to feed their other children, too, they are often willing to sell one of them to help the rest of the family stay alive. To combat human trafficking in Thailand, feeding programs can be a huge help. Feeding these children can eliminate the reason for selling them into slavery.

The most violent of all is the Albanian mafia. They lure their victims very often through alluring job offers into prostitution. They focus on women and they immediately begin drugging and raping them. They also keep law enforcement under threat, making it really hard to deal with them.

In the US model, pimps are looking for runaways, often making long-term relationships with them and helping them in the beginning, but later forcing their victims into prostitution. These victims also very often suffer from Stockholm syndrome. One newly emerging method to get new victims is the use of so-called Romeo pimps. These Romeo pimps are high school kids who are hired by real pimps. They would go to parties with these young girls and give them drugs there, and they would make pornographic videos of their victims and blackmail them with these videos into prostitution.

Human trafficking is also a very big problem in India, which is the most dangerous place to be a little girl. In India, besides children being sold for sex trafficking, they may often be mutilated to become beggars. Human trafficking in Mexico is run by international drug cartels, like the infamous MS-13, and this is just a part of their criminal portfolio, which includes money laundering and drug and gun trafficking. Finally, Nigeria and West Africa are very hard cases, where many people still believe that having sex with a young child can cure AIDS.

Lopez described a tool helpful against human trafficking called the Freedom App, where anyone experiencing anything related to human trafficking can send in information anonymously. This application already helped in freeing children held for prostitution. In addition, they have their own system that monitors all incoming information to help them with tracking and intervening in human trafficking.

An attendee asked how many false positives they get from the Freedom App. Lopez answered that they are not flooded by messages so far; in addition, before taking action, they contact local trusted officers who make sure that the information is correct. Someone asked what the community of computer security experts can do to battle human trafficking. Lopez answered that they need the most cutting-edge technology to stay ahead of criminals, but all help is welcome. Send an email with your expertise and they will help figure out how each individual can best help the cause. A questioner wondered how they determine who to trust among all the corrupt officers, especially in foreign countries. Lopez said they first contact people whom they trained in different techniques against criminals; second, they go to the local bureau to know who is trusted; most importantly, they can see whether a person is just working for the payroll or is passionate about helping victims and wants to bring justice to the criminals. For more information, go to www.orphansecure.com or write to info@orphansecure.com.

After Coffee Break Crypto

Summarized by Michael Zohner (michael.zohner@cased.de)

Burst ORAM: Minimizing ORAM Response Times for Bursty Access Patterns

Jonathan Dautrich, University of California, Riverside; Emil Stefanov, University of California, Berkeley; Elaine Shi, University of Maryland, College Park

Jonathan presented an Oblivious RAM (ORAM) system that is designed to provide quick responses under bursty workloads. When data is outsourced to the cloud, meta-information such as data access patterns can leak valuable information to the provider even when the data is encrypted. ORAM constructions were introduced to keep such access patterns hidden from malicious cloud providers. Existing ORAM constructions assume a steady stream of requests and primarily focus on minimizing the total bandwidth overhead, but real-world storage servers often have to cope with bursts of data queries. The presented Burst ORAM scheme was specifically designed to handle such bursts while minimizing response times of individual queries.

To cope with bursts of queries, Burst ORAM introduces several new techniques. First, it prioritizes the online I/O required for the client to obtain each result, delaying the more expensive shuffling I/O until idle periods between bursts. Second, it schedules shuffling jobs such that the most efficient jobs are prioritized. Third, it XORs blocks together before returning them to the client, reducing the online I/O to a constant amount.

The response times and bandwidth consumptions of Burst ORAM were simulated and compared to an insecure baseline system as well as a traditional ORAM system. The results showed that under a realistic workload with bursts of moderate lengths, Burst ORAM achieved response times that were comparable to the insecure baseline and orders of magnitude lower than traditional ORAM systems. However, Burst ORAM increases the total communication compared to traditional ORAM systems by up to 50%. Thus, a question to tackle in future work is how to reduce the overall communication overhead while keeping response times low.

Following the presentation, an audience member asked what would happen during an extremely large burst. Jonathan responded that in this case the costs of Burst ORAM would gracefully degrade toward those of traditional ORAM systems.

TRUESET: Faster Verifiable Set Computations

Ahmed E. Kosba, University of Maryland; Dimitrios Papadopoulos, Boston University; Charalampos Papamanthou, Mahmoud F. Sayed, and Elaine Shi, University of Maryland; Nikos Triandopoulos, RSA Laboratories and Boston University

Ahmed presented their work on TRUESET, a prototype for set-centric Verifiable Computation (VC) operations for outsourcing computations to a cloud. Next to privacy concerns, outsourcing work to an untrusted cloud server raises integrity and correctness concerns about computations done by the server. Verifiable computation was introduced to enable the client to verify the

correctness of a computation outsourced to a remote untrusted server. Research on VC has progressed in recent years both in terms of theory and practice. However, while current schemes maintain short proofs and short verification time, the proof computation time for the server is still too high to be considered usable. This is especially the case for set-centric operations used for database queries.

TRUESET was designed to reduce the proof computation time for the server and achieve an input-specific runtime while retaining the expressiveness of previous techniques. TRUESET achieves this by representing its operations using polynomials instead of arithmetic or Boolean circuits as done by traditional approaches. The polynomial circuit encodes the input size as a degree of the polynomial. Thereby, the circuit size becomes constant and independent of the size of the sets. Furthermore, special transformation gates can be used to transform the polynomial representation of sets into an arithmetic representation if additional non-set-centric operations are required.

TRUESET was implemented and compared to current systems for VC. Most prominently, TRUESET achieved more than 30x speed-up for the proof computation runtime compared to existing approaches for sets with more than 64 elements, reaching 150x speed-up for a union of two 256-element sets. Furthermore, TRUESET was shown to be applicable to sets with 30x larger size than previous approaches. Despite the large speed-up, Ahmed pointed out that their implementation was not yet practical, but is meant to spawn interest in practical VC systems.

Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture

Eli Ben-Sasson, Technion—Israel Institute of Technology; Alessandro Chiesa, Massachusetts Institute of Technology; Eran Tromer, Tel Aviv University; Madars Virza, Massachusetts Institute of Technology

Madars presented work on generating non-interactive zero-knowledge proofs that are short and easy to verify and rely on a setup that is independent of the proven function. When computing on decentralized information, we often encounter the problem that the goals of integrity and confidentiality are complementary to each other. For instance, if a server holds a confidential database and a client wants to evaluate a public function on his public input and the database, the client either has to trust the server to compute the correct output or the server has to disclose his database to the client. To enable this computation while bridging the gap between integrity and confidentiality, zero-knowledge proofs can be used. A zero-knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) is a special type of zero-knowledge system that builds short and easy-to-verify non-interactive zero-knowledge proofs that are based on suitable cryptographic assumptions. While practically feasible zk-SNARKs for certain applications exist, they rely on a costly trusted setup that is tailored to the evaluated function. Furthermore, they have only limited support for high-level languages.

Instead of being targeted to a particular function, Madars introduced a zk-SNARK system with universal setup that uses universal circuits to compute a proof for the desired functionality. The system takes as input bounds on the program size, the input size, and the time, and the corresponding trusted setup allows supporting all program computations that are within these bounds. However, in prior work, generating a universal circuit incurs a multiplicative overhead in the program size and is therefore only feasible for small programs. To allow the evaluation of larger programs, a routing network was introduced, which reduced the multiplicative to an (essentially) additive overhead.

The above circuit generator is composed with a zk-SNARK system for circuits. Both components were implemented and their performance was compared to existing approaches. As for the zk-SNARK for circuits, when evaluated on a one million-gate circuit with a thousand-bit input, its proof generation time was shown to be 5.3x faster and its proof verification time was shown to be 1.8x faster. The implementation was used in Zerocash, a privacy-preserving digital currency. Madars and his coauthors are looking for further applications of their work.

An audience member asked whether the authors had looked at universal circuit generators in the literature instead of constructing a new one. Madars replied that one could consider such computational models, but as their goal was SNARKs for RAM computations, it would incur a sub-optimal intermediate reduction: from RAM to circuits, followed by a universal circuit for circuits; they chose to have universal circuits that directly support RAM computations.

Faster Private Set Intersection Based on OT Extension

Benny Pinkas, Bar-Ilan University; Thomas Schneider and Michael Zohner, Technische Universität Darmstadt

Michael surveyed and optimized existing protocols for Private Set Intersection (PSI) in the semi-honest model and presented a novel protocol based on Oblivious Transfer (OT). PSI considers the problem of two parties each holding a set of elements and wanting to identify the elements they have in common without disclosing any other element. PSI protocols can, for instance, be used for secure database joins or discovery of common contacts. Since PSI is a general problem and is often used as an indicator for the practicality of secure computation, it has gained a lot of attention, and many protocols based on different techniques have been proposed. However, there have been various inconsistencies in the evaluation and comparison of the protocols, resulting in uncertainty about the best performing protocol for a particular deployment scenario.

Major results on PSI protocols were outlined and categorized depending on their underlying technique as public-key-based, generic secure-computation, or circuit-based and OT-based. Using current state-of-the-art techniques in secure computation, optimizations for a circuit-based and an OT-based protocol were proposed, which decreased both their runtime and communication by at least a factor of 2. Michael then introduced a new

OT-based PSI protocol that made use of recent improvements for OT extension and used hashing schemes to achieve better efficiency.

Finally, the performance of all surveyed protocols was evaluated using the same programming language, techniques, libraries, and benchmarking environment. The results showed that the public-key-based protocols had a moderate runtime for long-term security but had the most efficient communication complexity. The circuit-based protocols had the highest runtime and communication complexity but could be extended to other functionalities without requiring a proof-of-security. The OT-based protocols achieved the best overall runtime. To evaluate the practical usability of PSI, the results were compared to the performance of the naive hashing solution that is currently used in practice but that leaks information about the inputs. Compared to the best performing PSI protocol, the naive solution had an order of magnitude less runtime and communication.

The first question concerned the possibility of authenticating the input elements that are used by each party. Michael replied that although his work did not ensure authenticity of input elements, existing approaches could be used to achieve this property. The second questioner wanted to know how the results of the paper “Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?” in NDSS ’12 fit in with the overall results of the presented work. Michael replied that the results of the NDSS ’12 paper were later revised in the paper “Experimenting with Fast Private Set Intersection” in TRUST ’12, leading to confusion about the performance of the analyzed schemes.

Program Analysis: Attack of the Codes

Summarized by Brendan Saltaformaggio (bdsaltaformaggio@gmail.com)

Dynamic Hooks: Hiding Control Flow Changes within Non-Control Data

Sebastian Vogl, Technische Universität München; Robert Gawlik and Behrad Garmany, Ruhr-University Bochum; Thomas Kittel, Jonas Pfoh, and Claudia Eckert, Technische Universität München; Thorsten Holz, Ruhr-University Bochum

Sebastian Vogl presented a new approach, called dynamic hooks, to construct malicious code hooks residing purely in non-control data. As a running example, the Linux kernel’s `list_del` function was used to illustrate a dynamic hook. By crafting a special list node to be deleted, an attacker could cause an overwrite of a kernel return address or other control flow data. Using a combination of program slicing and symbolic execution, the researchers revealed 566 and 379 execution paths in the Linux and Windows kernels, respectively, that are vulnerable to such attacks.

The authors’ choice to use VEX was questioned. Vogl responded that they had tested others, but VEX worked best without any practical problems. William Enck, the session chair, asked how common are side effects in different paths. Vogl admitted that it requires an expert to look at a specific exploit path to determine whether that specific path works for a given attack.

X-Force: Force-Executing Binary Programs for Security Applications

Fei Peng, Zhui Deng, Xiangyu Zhang, and Dongyan Xu, Purdue University; Zhiqiang Lin, The University of Texas at Dallas; Zhendong Su, University of California, Davis

Fei Peng began his presentation citing the limitations of current binary analysis frameworks: Static analysis suffers from over-approximation and lack of runtime data. Dynamic analysis lacks coverage. Symbolic execution may not scale. Given these limitations, X-Force attempts to force a binary to execute as many code paths as possible by dynamically flipping select conditional branches.

Peng argued that, while X-Force is not complete or sound, it overcomes the limitations of current binary analysis platforms. X-Force is therefore designed as a new platform that binary analysis tools can be built upon. The presentation showed how X-Force could achieve far better CFG construction and indirect call coverage on most of the SPEC test suite binaries, compared to traditional static or dynamic analysis. Finally, Peng showed how an existing type reverse engineering tool (REWARDS) was ported to X-Force, and this caused a considerable increase in variable coverage.

The first questioner commented that X-Force is similar to concolic execution systems except that X-Force chooses to execute invalid paths to avoid the slowdown of concolic execution, and asked whether they had compared X-Force with any concolic execution systems. Peng replied that they did compare X-Force with the S2E system and that the results are in the paper. Peng was then asked whether X-Force can utilize parallelization speed-up runs for online use. His response was that X-Force already can parallelize multiple runs.

ByteWeight: Learning to Recognize Functions in Binary Code

Tiffany Bao, Jonathan Burket, and Maverick Woo, Carnegie Mellon University; Rafael Turner, University of Chicago; David Brumley, Carnegie Mellon University

Given the importance and difficulty of automatically identifying functions within binaries, Tiffany Bao presented a new solution called ByteWeight. For motivation, Bao showed how different levels of compiler optimizations break many common function boundary signatures. Further, she explained that the static binary analysis tool IDA is often unable to uncover functions at such high levels of optimization.

To address the function identification challenge, Bao described how ByteWeight combines machine learning and program analysis to perform function identification. First, training binaries are used to extract common function prefix sequences and build a weighted prefix tree. These weighted prefix trees are then used to identify function boundaries within test binaries.

Bao then presented the results of applying ByteWeight to 2200 binaries. Comparing against a previous approach by Rosenblum et al., ByteWeight uncovered far more functions within the test binaries. When compared to other function start identification

tools, ByteWeight's precision and recall was again higher. Lastly, Bao invited others to test the ByteWeight system by downloading a preconfigured VM at <http://security.ece.cmu.edu/byteweight/>.

Bao was asked why they chose to use the BAP platform over an IR. She responded that BAP was chosen because much of their implementation is based on BAP and that ByteWeight could also be ported to use other platforms. Eric Eide (University of Utah) noted that the paper's experiments are not compiler-specific and asked whether the results would be better if ByteWeight was trained on a specific compiler. Bao replied that the difference may not be large and that they do not need compiler-specific knowledge. Finally, a questioner asked how many prologues were in a common weighted prologue tree. Bao responded that in their current results, trees often contain thousands of nodes.

Optimizing Seed Selection for Fuzzing

Alexandre Rebert, Carnegie Mellon University and ForAllSecure; Sang Kil Cha and Thanassis Avgerinos, Carnegie Mellon University; Jonathan Foote and David Warren, Software Engineering Institute CERT; Gustavo Grieco, Centro Internacional Franco Argentino de Ciencias de la Información y de Sistemas (CIFASIS) and Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET); David Brumley, Carnegie Mellon University

With the popularity of fuzzing as a software testing technique, Sang Kil Cha presented a new effort to mathematically reason about how to pick the best seeds for fuzzing. "Best" here is finding the most bugs with specific fuzzing seeds. The presentation then explained the several different selection algorithms developed in the paper and how each of these algorithms compared to one another. Finally, the results of fuzzing a variety of applications with different seed-selection algorithms is shown, and the effectiveness of each algorithm is compared.

The session chair, William Enck, noted that the selection of seeds assumes that you have a base data set and asked how it handles applications that modify the fuzzed data. Sang Kil Cha responded that the current framework already handles such scenarios and that this does not affect their results.

After Lunch Break Crypto

Summarized by Michael Zohner (michael.zohner@cased.de)

LibFTE: A Toolkit for Constructing Practical, Format-Abiding Encryption Schemes

Daniel Luchaup, University of Wisconsin—Madison; Kevin P. Dyer, Portland State University; Somesh Jha and Thomas Ristenpart, University of Wisconsin—Madison; Thomas Shrimpton, Portland State University

Kevin presented LibFTE, an easy-to-use toolkit for instantiating Format-Preserving Encryption (FPE) and Format-Transforming Encryption (FTE) schemes. Several applications, such as in-place encryption of credit card numbers in a database, require that plaintexts and ciphertexts abide by a specific format. A related problem, which was investigated by the authors in their previous work "Protocol Misidentification Made Easy with Format-Transforming Encryption" at CCS '13, focuses on circumventing network censors by using FTE to transform ciphertexts into messages that are indistinguishable from real network protocol messages. Kevin revisited the notion of FTE and outlined how to instantiate an FTE scheme.

While FPE/FTE schemes exist, instantiating them for a particular format requires expert knowledge as well as engineering expertise to achieve good performance. LibFTE was designed to reduce the challenges in building FPE/FTE schemes. LibFTE allows the developer to specify the format of the plaintext and the format of the ciphertext separately using regular expressions. However, using prior techniques, to instantiate a FPE/FTE scheme, the regular expressions have to be transformed to a deterministic finite automaton (DFA), which can grow exponentially in size and can even require around 200 MB for some formats. To reduce the memory requirements for instantiating FPE/FTE schemes, the concept of relaxed ranking was introduced. In relaxed ranking, FPE/FTE encryption is performed directly from the nondeterministic finite-state automaton (NFA) representation of the regular language, which obviates the need for NFA to DFA conversion.

LibFTE has interfaces for C++, Python, and JavaScript and comes with a configuration assistant for instantiating FPE/FTE schemes. LibFTE was evaluated in several scenarios. Firstly, it was tested on 3,500 regular expressions from the Snort IDS, where it was able to instantiate FPE/FTE schemes for all expressions using less than 150 MB memory, even when FPE/FTE schemes without relaxed ranking failed. In addition, the average required memory was reduced by 30%. Secondly, its performance for simultaneous compression+encryption was compared to regular AES encryption using PostgreSQL, where it saved 35% disk space on average while maintaining a similar query time. Thirdly, a LibFTE Firefox extension was implemented and used to encrypt a Yahoo! address book contact form. LibFTE is available online at <https://libfte.org>.

Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens

Daniel Demmler, Thomas Schneider, and Michael Zohner, Technische Universität Darmstadt

Daniel presented his work on practical secure computation on mobile devices using a smart card to achieve more efficiency. Mobile devices have become an important tool in modern society. They measure, collect, and store various data throughout the daily life of their user and are used to perform various tasks. This makes them an important target for privacy measures such as secure computation. Secure computation enables two parties to evaluate a public function on their private inputs without leaking any information about either party's input except what can be obtained from the result. While secure computation on desktop PCs is becoming increasingly practical, its runtime on resource-constrained mobile devices is still too high to be considered usable.

To increase performance of secure computation on mobile devices, Daniel presented a scheme that uses a smart card, located on one device, as trusted third party. The scheme is secure against passive adversaries, works in three phases, and offloads the bulk of the workload of the secure computation protocol to the smart card. In the first phase, called the init phase,

the smart card pre-generates the data required for the secure computation. The init phase can be performed independently by the device holding the smart card at any point in time: for instance, when the device is being charged. In the second phase, called the setup phase, the smart card securely transmits the helper data, generated in the init phase, over a secure channel to the partner device the secure computation protocol is executed with. In the third phase, called the online phase, the devices run the secure computation protocol and obtain the output.

The presented scheme was implemented, evaluated, and compared to related work on three example applications using an Android smartphone and an off-the-shelf smart card. In the first application, which considered privacy-preserving scheduling of a meeting, the smart-card-based scheme achieved 3x better runtime than existing non-smart-card-based schemes. In the second application, the scheduling functionality was extended by location-awareness, demonstrating that the computed function could easily be changed. The third application considered private set-intersection and showed that the smart-card-based scheme achieved a similar performance on smartphones as a non-smart-card aided secure computation protocol execution on desktop PCs. The evaluation demonstrated that secure computation on resource constrained mobile devices can indeed be practical when supported by a trusted hardware token.

Someone asked Daniel whether an extension of his scheme to stronger adversaries was possible. Daniel replied that an extension to malicious adversaries would be possible by also equipping the partner device with a hardware token and massaging the TinyOT protocol introduced in “A New Approach to Practical Active-Secure Two-Party Computation” at CRYPTO ’12.

ZØ: An Optimizing Distributing Zero-Knowledge Compiler

Matthew Fredrikson, University of Wisconsin—Madison; Benjamin Livshits, Microsoft Research

Matthew presented work on ZØ, a zero-knowledge protocol compiler that combines two existing zero-knowledge systems and translates from C# to distributed multi-tier code. To maintain a client’s privacy, several applications move functionality to the client and send only aggregated outputs to a server. However, the client can not necessarily be trusted to return the correct output to the server. Hence, there is a tradeoff between the privacy of the client and the integrity of the output. Google Waze was mentioned as a prominent example for this tradeoff, where users provide their traffic data to improve quality of routing information.

Zero-knowledge protocols are a common tool to fulfill the two complementary goals of privacy and integrity. To ease the development of zero-knowledge protocols, zero-knowledge protocol compilers, such as Pinocchio and ZQL, have been introduced. However, these compilers are directly based on one particular technique for generating zero-knowledge proofs and scale very poorly to large applications and to applications that cannot efficiently be expressed using the underlying technique.

The main focus of ZØ is to make the generation of efficient zero-knowledge protocols easier. ZØ compiles from C# and allows developers to specify zero-knowledge regions using LINQ syntax. To support code-generation for different distributed scenarios, ZØ allows splitting functionality among multiple tiers. Additionally, it generates more efficient zero-knowledge protocols that scale to larger problems by not tying itself to a particular zero-knowledge technique. This is done by combining both the techniques that are used in Pinocchio and ZQL and using a cost model to schedule the techniques across multiple tiers such that the resulting protocol obtains the most efficient runtime.

To evaluate the performance benefits of ZØ, six different real-world applications were implemented in ZØ, Pinocchio, and ZQL, and the performance of the resulting protocols was compared. Overall, ZØ allowed to scale up to 10x larger application sizes and resulted in up to 40x faster runtime compared to Pinocchio and ZQL.

SDDR: Light-Weight, Secure Mobile Encounters

Matthew Lentz, University of Maryland; Viktor Erdélyi and Paarijaat Aditya, Max Planck Institute for Software Systems (MPI-SWS); Elaine Shi, University of Maryland; Peter Druschel, Max Planck Institute for Software Systems (MPI-SWS); Bobby Bhattacharjee, University of Maryland

Matthew outlined the Secure Device Discovery and Recognition (SDDR) system for short-range secure mobile encounters. Many mobile social services, such as Huggle and Foursquare, detect nearby peers (strangers and/or friends) and support communication among these peers. Traditional approaches that solve this problem either use a centralized service or a decentralized approach relying on device-to-device communication with static addresses (or IDs), which both allow tracking users’ movements. Using random IDs, on the other hand, prevents the device from being recognizable by friendly devices.

SDDR avoids tracking while allowing the recognition of friendly devices by using a decentralized approach in combination with random user IDs and cryptographic techniques. Existing cryptographic techniques, however, perform poorly on resource-restricted devices such as mobile phones, both in terms of runtime and energy consumption. Matthew outlined a novel non-interactive solution that uses the Bluetooth controller to send a beacon message upon query. This beacon enables secure communication between peers and can allow permitted friends to recognize the device. In addition to granting people the right to recognize a device, SDDR also introduces methods to efficiently revoke this permission.

SDDR was implemented on an Android device, and its runtime was measured as well as its energy consumption. The runtime for a single query amounted to less than one millisecond, four orders of magnitude faster than the protocol that uses existing cryptographic techniques. The energy consumption was measured over a period of 30 minutes and extrapolated to the total consumption for a day. While SDDR consumed around 10% of the phone’s battery capacity, the protocol based on existing cryptographic techniques required 191% (depleting the battery in half a

day). As an example for further applications, such as facilitating communication among groups of peers who encountered each other presently (or in the past), a reference to the authors' paper "EnCore: Private, Context-based Communication for Mobile Social Apps" was given. The code for SDDR is available online at <http://www.cs.umd.edu/projects/ebn/>.

Program Analysis: A New Hope

Summarized by Lucas Davi (lucas.davi@trust.cased.de)

Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM

Caroline Tice, Tom Roeder, and Peter Collingbourne, Google, Inc.; Stephen Checkoway, Johns Hopkins University; Úlfar Erlingsson, Luis Lozbo, and Geoff Pike, Google, Inc.

Caroline Tice presented a compiler-based Control-Flow Integrity (CFI) approach to prevent runtime attacks. Caroline argued that return addresses and stack data are today well-protected due to stack canaries and address space layout randomization. Hence, attackers typically corrupt vtable or function pointers to launch a runtime attack. To tackle these attacks, Caroline presented a compiler-based approach that enforces so-called forward-edge control-flow integrity (CFI) which instruments indirect calls and jumps with CFI checks. Specifically, she presented VTV (Virtual-Table Verification) for GCC, and IFCC (Indirect Function-Call Checks) for LLVM. The presented CFI solution is open source, induces a modest overhead of 1–8.7%, and protects 95–99.8% of all indirect function calls.

Lucas Davi (TU Darmstadt) asked whether forward-edge CFI prevents an adversary from exploiting an indirect call or jump to invoke VirtualAlloc or VirtualProtect in a modern application like Adobe Reader. Caroline responded that this depends on the virtual calls used by the target application. David Evans (University of Virginia) asked at which time forward-edge CFI will be applied to the Chrome browser. Caroline mentioned that this is the goal of the project, but there are still some issues to tackle.

ret2dir: Rethinking Kernel Isolation

Vasileios P. Kemerlis, Michalis Polychronakis, and Angelos D. Keromytis, Columbia University

Vasileios started with an introduction to kernel exploits and mentioned that most existing kernel attacks are based on exploiting a memory corruption vulnerability in the kernel to redirect execution to a shellcode or ROP payload residing in user space. Such attacks are referred to as ret2usr attacks and can be detected by new hardware and compiler-based defenses (SMEP, SMAP, PXN, KERNEXEC, UDEREF, and kGuard) that basically all prevent the kernel from either executing code or tampering with reference data from userland. However, Vasileios demonstrates that all these defenses can be bypassed with a new attack technique called return-to-direct-mapped memory (ret2dir). The main idea is to redirect execution to a kernel memory region called physmap that contains a direct mapping of all the physical memory in the system (including pages mapped from user space). Hence, the attacker only needs to know where his shellcode is mapped to in the physmap region to execute the shell-

code from the kernel space. To defend against ret2dir attacks, Vasileios presented exclusive page frame ownership (XPFO) that unmaps userland pages from physmap and remaps them (deleting the page contents) when they are reclaimed by user space.

David Evans (University of Virginia) asked whether sharing of data between kernel and user space is common in modern systems as this would bypass the presented defense against ret2dir attacks. Vasileios replied that it depends on how frequently page caching is used by the system. Someone asked whether existing kernel data integrity protection mechanisms would prevent ret2dir attacks. Vasileios confirmed that such integrity mechanisms would also defend ret2dir attacks. Someone asked whether these attacks were also possible in Windows. Vasileios replied that Windows also has a page cache and the attacks are not specific to Linux.

JIGSAW: Protecting Resource Access by Inferring Programmer Expectations

Hayawardh Vijayakumar and Xinyang Ge, The Pennsylvania State University; Mathias Payer, University of California, Berkeley; Trent Jaeger, The Pennsylvania State University

Hayawardh started with a motivating example to demonstrate that resource access control is an important problem. In particular, he showed how a university's Apache Web server can be compromised by a student (who owns a page on the Web server) to retrieve the password file exploiting a symbolic link to the password file. Hayawardh elaborated on two reasons why such attacks were still possible: (1) Programmers, administrators, and OS distributors have different expectations on how files are protected and used on the system, and (2) code complexity makes it challenging for a programmer to protect every resource used in his program. To tackle this security problem, Hayawardh presented a solution to map programmer expectation onto a system. In particular, Hayawardh presented a process firewall based on introspection of the program limits system calls to their appropriate resources as given by their original intent and expectation. As a motivation to deploy the process firewall, the authors' evaluation showed that for four of five tested programs, more than 55% of the resource accesses were not protected with defensive checks or filters.

Rob Johnson (Stony Brook) asked Hayawardh how policies were generated and about the advantages of using the process firewall compared to fixing the code. Hayawardh replied that the process firewall runs automated methods to generate the corresponding access control policies.

Static Detection of Second-Order Vulnerabilities in Web Applications

Johannes Dahse and Thorsten Holz, Ruhr-University Bochum

Facebook Internet Defense award!

Johannes presented a static analysis tool that detects second-order vulnerabilities in Web applications. He started by describing first-order vulnerabilities such as the well-known SQL

injection attack. These attacks can be prevented by applying sanitization. However, second-order vulnerabilities occur when an attack payload is first persistently stored in a database or file, and the application reads in a second stage the payload to perform a security-critical operation. To identify these vulnerabilities, Johannes presented a static source code analysis tool (focusing on PHP) that analyzes write and read operations of an application to persistent data. In general, the static analysis tool builds a control-flow graph of the application, identifies sensitive sinks, and validates whether user input can potentially write to a sensitive sink recording and considering also possible sanitization on data inputs. The same is also done for inputs that originate from persistent data storage to cover read operations by the application. Finally, data input writes and reads are correlated with each other to connect input and output points to identify second-order vulnerabilities. The evaluation of six Web applications showed that the static analysis approach identified 159 true positives (second-order vulnerabilities) and 43 false positives.

Benjamin Livshits (MSR) asked Johannes about the lessons learned, and why the analysis did not take other languages beyond PHP into account. Johannes replied that in contrast to Java or other languages, PHP is particularly vulnerable to second-order vulnerabilities. However, it was still possible to write PHP-secure code. Someone mentioned that static analysis has limitations because it misses dynamic behavior. Johannes noted that there were some false positives and that the presented approach focuses on vulnerabilities that can be detected at static analysis time.

Mobile Apps and Smart Phones

Summarized by Shouling Ji (sji@gatech.edu)

ASM: A Programmable Interface for Extending Android Security

Stephan Heuser, Intel CRI-SC at Technische Universität Darmstadt; Adwait Nadkarni and William Enck, North Carolina State University; Ahmad-Reza Sadeghi, Technische Universität Darmstadt and Center for Advanced Security Research Darmstadt (CASED)

Stephan Heuser presented work on promoting OS security extensibility in the Android OS. Specifically, the authors designed a framework named Android Security Modules (ASM) that provides a programmable interface for defining new reference monitors for Android. In the presentation, Heuser first demonstrated the architecture of Android, which consists of three layers (from top to bottom): the App layer, the Android OS layer, and the Linux kernel layer. Currently, to improve the security of the Android system, access control is implemented in every layer. Subsequently, to motivate their work, Heuser summarized over a dozen recent Android security architecture proposals to identify the hook semantics requirements for Android security models. Based on their survey, they concluded that Android OS is responsible for enforcing more than just UNIX system calls. They also identified that it is necessary for authorization hooks to replace data values and for third-party applications to introduce new authorization hooks.

Based on their findings, they designed and implemented an extensible Android Security Modules (ASM) framework. Heuser introduced how ASM works layer by layer. Basically, ASM allows multiple simultaneous ASM apps to enforce security requirements while minimizing the system overhead. To demonstrate the utility and performance of the proposed ASM framework, they implemented several ASM apps. In the presentation, Heuser showed one ASM app MockDroid, which is a system-centric security extension for the Android OS, allowing users to gracefully revoke the privileges requested by an application without the app caching. Besides that, Heuser also demonstrated the performance overhead and energy consumption of the ASM framework.

Following Heuser's talk, there was an interesting discussion on the ASM framework. First, someone was curious about the overall design and the novelty of ASM, especially the work mechanism of ASM in the kernel layer. Heuser summarized the design of ASM and highlighted the distinguished features of ASM. He also directed the audience to find more discussion in the paper. Another attendee was curious about how ASM implements the isolation of apps. Based on Heuser's response, ASM does not separate apps (or components) in its current version. It is an interesting future research direction of this paper.

Brahmastra: Driving Apps to Test the Security of Third-Party Components

Ravi Boraskar, Microsoft Research and University of Washington; Seungyeop Han, University of Washington; Jinseong Jeon, University of Maryland, College Park; Tanzirul Azim, University of California, Riverside; Shuo Chen, Jaeyeon Jung, Suman Nath, and Rui Wang, Microsoft Research; David Wetherall, University of Washington

Jaeyeon Jung presented their solution to the problem of third-party component integration testing at scale, in which one party wishes to test a large number of applications using the same third-party component for a potential vulnerability. First, Jung analyzed the status quo of the use of third-party components and why they are commonly used. Subsequently, Jung pointed out that the use of third-party components may cause some security risks, which have been demonstrated in several existing reports. Aiming at understanding the security of third-party components, they designed an app automation tool named Brahmastra for helping app stores and security researchers test third-party components in mobile apps at runtime.

Jung motivated their design by analyzing the limitations of existing third-party component testing tools. Taking Monkey as an example, Jung showed its vulnerability step by step using a demo in which Monkey leaked people's Facebook profiles. Then Jung presented their approach, which leverages the structure of Android apps to improve test hit rate and execution speed. The core techniques of their approach include two aspects: They characterize an app by statically building a page transition graph and call chains, and they rewrite the app under test to directly invoke the callback functions that trigger the desired page transitions.

Jung also showed the design and implementation of their testing tool Brahmastra. To evaluate the performance of Brahmastra, they tested 1010 popular apps crawled from Play Store. According to their results, Brahmastra significantly outperforms the existing solution PUMA with respect to the hit rate and test speed. Finally, Jung also demonstrated their security analysis in two scenarios: ads in kids' apps and social media add-ons. Based on their testing results, 175 out of 220 children's apps point to Web pages that attempt to collect personal information, which is a potential violation of the Children's Online Privacy Protection Act (COPPA); and 13 of the 200 apps with the Facebook SDK are vulnerable to a known access token attack.

Someone pointed out that Brahmastra is a goal-driven tool. It might be unfair to compare it with PUMA. Jung responded that Brahmastra is designed to help app stores and security researchers test third-party components in mobile apps at runtime. Therefore, both Brahmastra and PUMA have their advantages and disadvantages. Another person asked about the code crush of Brahmastra. Jung pointed out that if the app resists being rewritten, it is possible that the program is crushed.

Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks

Qi Alfred Chen, University of Michigan; Zhiyun Qian, NEC Laboratories America; Z. Morley Mao, University of Michigan

Qi Alfred Chen explained that on the Android system, a weaker form of GUI confidentiality can be breached in the form of UI state by a background app without requiring any permissions. First, Chen explained the importance of GUI security. Since GUI content confidentiality and integrity are critical for end-to-end security, the security of smartphone GUI frameworks remains an important topic. Then, Chen showed a weaker form of GUI confidentiality breach, which might enable UI state hijacking. Through a demo, Chen demonstrated how UI state hijacking attack steals people's passwords in the H&R Block app. In addition, Chen also showed that this can enable other attacks, which can be classified as UI state inference attacks.

Chen summarized the underlying causes of such UI state inference attacks. This is mainly because the Android GUI framework design leaks UI state changes through a publicly accessible side channel, which is a newly discovered shared-memory side channel. Based on this finding, Chen showed the general steps of UI state inference attacks, which mainly consist of three steps: activity transition detection; activity inference; and UI state hijacking, camera peeking, and other UI state inference attacks. To detect the activity transition, the newly discovered shared-memory side channel will be employed. For activity inference, besides the newly discovered shared-memory side channel, other side channels, e.g., CPU, network activity, will also be used. Finally, they also evaluated the UI state inference attacks on seven popular Android apps, including WebMD, Chase, Amazon, Newegg, Gmail, and H&R Block. The results show that for six of the seven apps, the UI state inference accuracies are 80–90%

for the first candidate UI states, and over 93% for the top three candidates.

Following Chen's talk, several concerns were raised. First, someone was curious about whether the attacks were reported to Google. Chen responded that the system is still under testing. They will report the vulnerability to Google after finishing all the tests. Another attendee asked whether the presented attacks were device-specific. Chen said they tested the attacks on popular devices and it should be easier to implement the attacks on new devices.

Gyrophone: Recognizing Speech from Gyroscope Signals

Yan Michalevsky and Dan Boneh, Stanford University; Gabi Nakibly, National Research & Simulation Center, Rafael Ltd.

Yan Michalevsky presented a new attack for recognizing speech from gyroscope signals generated from iOS and Android phones. In the talk, Michalevsky first introduced how a MEMS gyroscope works and presented the initial investigation of its properties as a microphone. Subsequently, Michalevsky showed that the acoustic signal is sufficient to extract information of the speech signal, e.g., speaker characteristics and identity. The extraction leverages the fact that aliasing causes information leaks from higher frequency bands into the sub-Nyquist range. Third, Michalevsky demonstrated that isolated word recognition can be improved if the gyroscopes of multiple devices that are in close proximity can be sampled. They also evaluated their approach by repeating the speaker-dependent word recognition experiment on signals reconstructed from readings of two Nexus 4 devices. Finally, several suggestions were made to mitigate the potential risks of such an attack.

A questioner wondered about the countermeasures of limiting the sensors/meters of smartphones. Michalevsky summarized present solutions and proposed possible future research directions. An attendee pointed out that some designs have been proposed recently to prevent apps from reading (critical) sensor readings. Michalevsky responded that even so, there are still a lot of apps that have such specific permissions to access sensor readings. Therefore, such attacks are possible and very likely to happen in the real world. Finally, a questioner wondered whether the authors considered other sensor/meter readings. Michalevsky confirmed that they also looked at other sensor/meter readings. In this paper, however, they focused on recognizing speech from gyroscope signals.

Panel

The Future of Crypto: Getting from Here to Guarantees

The summary of this session is available online as an electronic supplement: www.usenix.org/login/dec14.

The complete summaries from CSET '14, 3GSE '14, FOCI '14, HealthTech '14, and WOOT '14 are available online as electronic supplements: www.usenix.org/login/dec14.

REAL SOLUTIONS FOR REAL NETWORKS

FREE DVD

zentyal Small Business Server
64-bit Community Edition 3.5

10 Tips for Better SSL

ADMIN
Network & Security

FREE DVD!

ADMIN

Network & Security

DEC 2014/JAN 2015

10 Tips for Better SSL

Best practices for managing website risk

PORT KNOCKING
Don't miss this cool trick for securing remote access

Ceph or GlusterFS?
Choosing a shared storage alternative

Free AD Tools
Manage your Active Directory on a budget

Top Tools

FIDO Alliance
Exploring the future of authentication

• Unix • Solaris

Log Parser Studio
Reading logfiles on Windows networks

zine.com

ADMIN
DEC/JAN 2015
US\$ 15.99
CAN\$ 17.99
01



0 74470 86640 4



Each issue delivers technical solutions to the real-world problems you face every day.

Learn the latest techniques for better:

- network security
- system management
- troubleshooting
- performance tuning
- virtualization
- cloud computing

on Windows, Linux, Solaris, and popular varieties of Unix.

**FREE
CD or DVD
in Every Issue!**

6 issues per year!

ORDER ONLINE AT: shop.linuxnewmedia.com



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

Save the Date

February 16–19, 2015 • Santa Clara, CA

FAST¹'15

**13th USENIX Conference
on File and Storage
Technologies**

The 13th USENIX Conference on File and Storage Technologies (FAST '15) brings together storage-system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems. The conference will consist of technical presentations, including refereed papers, Work-in-Progress (WiP) reports, poster sessions, and tutorials.

JUST ANNOUNCED! The FAST '15 Keynote Presentation will be given by Dr. Marshall Kirk McKusick.

Full program and registration information will be available soon.

www.usenix.org/fast15