;login:

DECEMBER 2013 VOL. 38, NO. 6

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Security

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

FAST '14: 12th USENIX Conference on File and Storage Technologies
February 17–20, 2014, Santa Clara, CA, USA
www.usenix.org/conference/fast14

2014 USENIX Research in Linux File and Storage Technologies Summit
In conjunction with FAST '14
February 20, 2014, Mountain View, CA, USA
Submissions due: January 17, 2014

NSDI '14: 11th USENIX Symposium on Networked Systems Design and Implementation
April 2–4, 2014, Seattle, WA, USA
www.usenix.org/conference/nsdi14

2014 USENIX Federated Conferences Week
June 17–20, 2014, Philadelphia, PA, USA

USENIX ATC '14: 2014 USENIX Annual Technical Conference
www.usenix.org/conference/atc14
Paper titles and abstracts due January 28, 2014

HotCloud '14: 6th USENIX Workshop on Hot Topics in Cloud Computing
www.usenix.org/conference/hotcloud14

WiAC '14: 2014 USENIX Women in Advanced Computing Summit

HotStorage '14: 6th USENIX Workshop on Hot Topics in Storage and File Systems
www.usenix.org/conference/hotstorage14
Submissions due: March 13, 2014

UCMS '14: 2014 USENIX Configuration Management Summit

ICAC '14: 11th International Conference on Autonomic Computing

USRE '14: 2014 USENIX Summit on Release Engineering

23rd USENIX Security Symposium
August 20–22, 2014, San Diego, CA, USA
www.usenix.org/conference/usenixsecurity14
Submissions due: Thursday, February 27, 2014

Workshops Co-located with USENIX Security '14

EVT/WOTE '14: 2014 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections

USENIX Journal of Election Technology and Systems (JETS)
Published in conjunction with EVT/WOTE
www.usenix.org/jets
Submissions for Volume 2, Issue 3, due: April 8, 2014

HotSec '14: 2014 USENIX Summit on Hot Topics in Security

FOCI '14: 4th USENIX Workshop on Free and Open Communications on the Internet

HealthTech '14: 2014 USENIX Workshop on Health Information Technologies
Safety, Security, Privacy, and Interoperability of Health Information Technologies

CSET '14: 7th Workshop on Cyber Security Experimentation and Test

WOOT '14: 8th USENIX Workshop on Offensive Technologies

OSDI '14: 11th USENIX Symposium on Operating Systems Design and Implementation
October 6–8, 2014, Broomfield, CO, USA
www.usenix.org/conference/osdi14
Abstract registration due April 24, 2014

Co-located with OSDI '14:

Diversity '14: 2014 Workshop on Diversity in Systems Research

LISA '14: 28th Large Installation System Administration Conference
November 9–14, 2014, Seattle, WA, USA
https://www.usenix.org/conference/lisa14
Submissions due: April 14, 2014

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 2013 VOL. 38, NO. 6

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Rikki Endsley
rikki@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Arnold Gatilao
Casey Henderson
Michele Nelson

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.

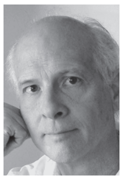
$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for non-members are $90 per year. Periodicals postage paid at Berkeley, CA, and additional offices.

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

EDITORIAL

Musings

RIK FARROW



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

As often is the case, I find myself musing about the state of computer security. Although there certainly is no easy answer to fixing our insecure systems, I've come across a wonderful analogy, thanks to a NOVA (US public television science) show: "Why Ships Sink" [1].

For ships at sea, as well as airplanes, the answer is often simple: human error is at least partially to blame. But nothing is as simple as it may first appear.

Bulkheads

By the time the *Titanic* sailed, ship designers included bulkheads in their designs. These bulkheads separated the region below the waterline of a ship into separate compartments. The goal for these compartments was to limit flooding if two ships collided. The bow of most ships also included a separate compartment, the *peak tank* that was designed to both crumple and contain any flooding from a collision.

As we all know, *Titanic*'s bulkheads failed rather dramatically. Instead of taking days to sink, *Titanic* took just hours [2]. The bulkheads were not actually watertight but could be, and were, overtopped by flooding. And these compartments were designed under the assumption that a ship would be holed in, at most, two compartments, and then only if a collision occurred right at the boundary between the two compartments.

The sinking of the *Oceanos* [3] provides another vivid example of the failure of watertight bulkheads. Ship designers had done a much better job by this time, having learned from the *Titanic*'s failure. But humans could easily foil this design. In the case of *Oceanos*, partially completed maintenance allowed a leak that started in the engine room to pass through a hole in a bulkhead into the sewage waste disposal tank, and from there, into the rest of the ship via toilets, sinks, and showers. A check valve that would have stopped the backwards passage of water through the waste lines had been removed and not replaced, leading to the sinking of the ship in rough seas off the coast of South Africa.

I certainly find it interesting how both of these examples included assumptions in design and compounded them with the actions of humans.

Computer Security

We have bulkheads, of a sort, in most of our computer systems. Memory management separates access to the memory of one process by other processes. And there are "rings" of protection, with the kernel running in the innermost ring, any VMs and possibly device drivers running in the next one or two levels, and user processes running in the outermost ring [4]. Hardware enforces these rings, so we can imagine them functioning as bulkheads within our computer systems, designed to prevent exploitation, rather than flooding. Attacks at the outermost ring should not impact inner rings.

Like the *Titanic*, which had doors in its bulkheads, CPUs also have "doors" between rings. These provide access to privileged routines—for example, allowing an editor to access blocks on a disk or a Web browser to read data from a network connection. We call these doorways *system calls*, but at the hardware layer they are software interrupts that cause execution to switch between the application that executed the interrupt and the interrupt or trap handler at an inner ring. This handler uses a value to index into the table of system calls.

System calls provide an entry point into the inner rings, and the innermost ring has access to all memory and all hardware. Behind the system call is what provides the weakness in the design: the operating system itself. Operating systems are giant concurrent programs that have several important features: they are crucial for the proper functioning of systems, they are large, and they are difficult to write.

As it turns out, kernel exploits have become one of the most common methods for escalating privilege on *nix systems. In the pre-Internet days, set-user-id (SUID) programs owned by the root were the most popular means for privilege escalation. As the Internet became widely used, root-run network services became more popular targets. And later, largely because of the awareness that both SUID root and root-run network services were dangerous, the numbers of both have decreased over time. There still are many SUID root programs, though not as many as there once were. And the number of root-run network services has declined dramatically. Also, kernel developers have designed kernel-based mechanisms, such as capabilities and SELinux, that can limit the scope of what SUID programs and network services are permitted to do.

That leaves the kernel as a huge program with a complete set of privileges and no limitations. Any code running at ring 0 has complete access to the system, making the kernel a juicy target.

According to conversations with people who run lots of Linux systems, the usual path to exploitation is to gain access to a system through theft of an account, then to use a kernel exploit to gain total control over the system in question. Often, the next step is to install trojan SSH/SSHD programs, so the attacker can steal more accounts.

Our watertight bulkheads are no more watertight, or better designed, than the *Titanic*'s.

The Lineup

We start off this security-focused issue with an article by Jon Howell and friends. Jon and his cohorts have published two papers about Embassies, and after some badgering they have completed an article about their new notion of how Web browsers should work. Instead of building many different browsers that are more like operating systems with lots of leaky bulkheads, they have built a system that runs complete binary applications within a Web browser. Unlike systems such as XaX [5] and Native Client [6] that came before them, Embassies does not require extensive code revisions in applications. Instead, Embassies does something I imagined (and wrote about [7]) many years before. Embassies uses a special library as a replacement for libc and ntdll.dll that provides an extremely limited system call interface to applications. In essence, Embassies reduces the number of openings left in the bulkheads between applications and the kernel to less than ten, far from the hundreds (to thousands) of system calls found today.

Sarah Meiklejohn and her associates wrote about Bitcoin. In their research, they used bitcoins to make online purchases, and through analyzing information used in these transactions, were able to group a goodly fraction of all Bitcoin addresses to a number of well-known entities, such as Mt. Gox and Silk Road. Their work shows that bitcoin transactions are not as anonymous as you might think, and the authors do a great job of explaining both their research and how Bitcoin works.

I interviewed Ben Laurie because a friend had pointed out that he had strong views about Bitcoin. Of course, Ben spends most of his time working to make the Internet safer, through his current work on Certificate Transparency [8]. I did get to ask Ben for his thoughts about digital currencies in general, and Bitcoin in particular.

At Crispin Cowin's request, I asked Istvan Haller and his co-authors to write about their smart fuzzer. Crispin had just been awarded the Test of Time for his work on stack canaries, and he told me this was his favorite work at the 2013 Security conference. Haller et al. combine previous work into a technique that zeroes in on areas within programs that are the best places to find buffer overflows, which is still an issue after all these years.

Although people presented a lot of other exciting research during Security '13, I chose only one other workshop paper for this issue. Mohammad Karami and Damon McCoy had researched DDoS for hire, and presented a workshop paper about this during LEET. I found what they had uncovered fascinating: for a small monthly fee, you can have a service DDoS the IP address of your choice with up to hundreds of millions of packets per second.

Justin Troutman had long been promising me an article about a new framework for cryptography. He and Vincent Rijmen (best known for his part in developing AES) have been researching how best to build a cryptographic framework that works well for both developers and end users. Today, cryptographic APIs leave developers with too many choices to make, choices that should instead be made by cryptographers who understand the theory behind how cryptographic primitives should be used. And most cryptographic libraries result in programs that are difficult for end users to use properly. So instead of having developers making design mistakes to produce programs that end users cannot simply use correctly, Troutman and Rijmen's goal is to create a "green" framework that solves both of these problems.

Phil Pennock contacted me, after some urging by Doug Hughes, about problems he has with how people perceive PGP. Phil runs a keyserver and is a code committer, so he is well situated to comment on PGP. Phil tells us that PGP cannot do many of the things that people expect it to do, for example, prevent traffic analysis. In fact, PGP makes traffic analysis simpler. Phil goes on to explain things you need to know if you want to use PGP properly.

David Lang continues with his series of articles about enterprise logging. David explains how to use the Security Event Correlator, SEC, as a tool for monitoring logs. I had long found SEC a complicated and hard to understand tool, and am glad that David has taken the time to carefully explain how to use some of its most important features.

James Plank has written a survey of erasure codes for storage systems. Most of us are at least somewhat familiar with RAID, a system that in most configurations relies on erasure coding to create a more durable storage system. Jim has presented many papers about erasure coding at FAST workshops, and does a great job in this article of explaining the different ways erasure coding works, and how to measure the effectiveness of erasure coding. Although you might think this is a topic that you don't need to understand, you will understand both erasure coding and RAID much better if you do read his article.

David Blank-Edelman has written about the command line. Sound boring? Well, it's not, as David provides helpful Perl modules and information that makes it easy to parse command lines, and strongly suggests that you not go and build yet another wheel.

David Beazley explains Python packages and what takes the place of main() in Python scripts. I often wondered about this, and, as usual, David provides lots of clear examples of how to access functions within packages as if they were the entry function in a C program.

Dave Josephsen writes to us from the wilderness about his adventures. Well, he only wrote a little bit about hiking in the Rockies, and spent most of his column extolling the usefulness of Go. Dave has discovered that by programming in Go, he has been encouraged to use Git, add network interfaces, think about concurrency, and embrace types and data structures. That's pretty amazing for both a computer language and a curmudgeon like Dave.

Dan Geer and guest co-author Michael Roytman point out that using guesstimates of a vulnerability's likelihood of being exploited makes no sense at all. They share charts and data with us to prove that calculated measures of exploitability do not match up with the vulnerabilities actually exploited, and provide suggestions for doing a better job of deciding what is most vulnerable.

Robert Ferrell gets serious about security in his column. Not that he isn't still being funny, but Robert makes a number of very good points, similar to points I was hearing in hallway talk during the Security Symposium.

Elizabeth Zwicky reviewed five books this time, three on management and two on data analysis. Mark Lamourine reviewed three short books on Vagrant, Git, and Puppet types and providers.

We have many more pages of summaries than we can print, all from the 2013 Security Symposium and the workshops during that week. If you have ever wondered why some things get summarized and others don't, summarizing is a volunteer activity. We do ask any person who has received financial assistance to attend USENIX events to summarize, but we do not force them to summarize. We have learned from experience that the best summaries come from interested participants who have a desire to write summaries.

That said, the volunteers managed to cover all of the Symposium, HotSec, WOOT, LEET, and parts of CSET and HealthTech. We also strive to post the summaries to the *;login:* portion of the USENIX Web site as soon as they have been edited, copyedited, and typeset, and you will often be able to find summaries weeks before they appear in print.

One of the more interesting things I've heard recently about security (which doesn't seem that new at all) is that you don't need to wonder whether your systems will be exploited; you need to *notice* when they have been. If you read the October issue's "For Good Measure" column, you may recall that in the Verizon Data Breach Investigations Report, 80% of data breaches are discovered by some unrelated third party. Geer and Pareek also reported that 65% of the people they survey reported discovering an attack aimed at some other party.

Perhaps we need to quit worrying about the security of our systems, start monitoring for signs of a successful exploit, and keep our incident response teams ready for the emergency that might sink our already leaky ships. We don't have watertight bulkheads, but *Titanic*s cruising serenely along into a night sea scattered with icebergs.

References

[1] Nova on "Why Ships Sink": http://www.pbs.org/wgbh/nova/tech/why-ships-sink.html.

[2] Watertight compartments on the *Titanic:* http://www.titanic-titanic.com/titanic_watertight_compartments.shtml.

[3] Sinking of the *Oceanos:* http://en.wikipedia.org/wiki/MTS_Oceanos.

[4] Rings for computer security: http://arstechnica.com/security/2009/03/storm-over-intel-cpu-security-could-be-tempest-in-a-teapot/.

[5] John R. Douceur, Jeremy Elson, Jon Howell, and Jacob R. Lorch, "Leveraging Legacy Code to Deploy Desktop Applications on the Web," *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI),* December 2008, pp. 339–354.

[6] Native Client: http://code.google.com/p/nativeclient/.

[7] Rik Farrow, "Musings," *;login:*, vol. 32, no. 4, 2007: https://www.usenix.org/publications/login/august-2007-volume-32-number-4/musings.

[8] Certificate Transparency: http://www.certificate-transparency.org/.

SECURITY

The 10-Kilobyte Web Browser

JON HOWELL, BRYAN PARNO, AND JOHN R. DOUCEUR

Jon Howell is re-envisioning the delivery of client applications, building a large scale, consistent distributed name service, building formally verified software, and improving verifiable computation. howell@microsoft.com

Bryan Parno focuses on protocols for verifiable computation and zero-knowledge proofs, building practical, formally verified secure systems, and developing next-generation application models. parno@microsoft.com

John R. Douceur is interested in the design of distributed algorithms, data structures, and protocols, and in the measurement, evaluation, and analytical modeling of systems and networks, with particular focus on statistical analysis and simulation. douceur@microsoft.com

In theory, browsing the Web is safe: click a link, and if you don't like what you see, click "close" and it disappears forever. In practice, this guarantee doesn't hold, because the browser is complex in both implementation and specification. We designed and built an alternate Web app delivery model in which the client-side interface specification and code—the pieces that replace the browser—are extremely simple, yet can run applications even richer than today's JavaScript apps. This article describes how we achieve this goal, and suggests a path forward into a future free of today's bloated browser interface.

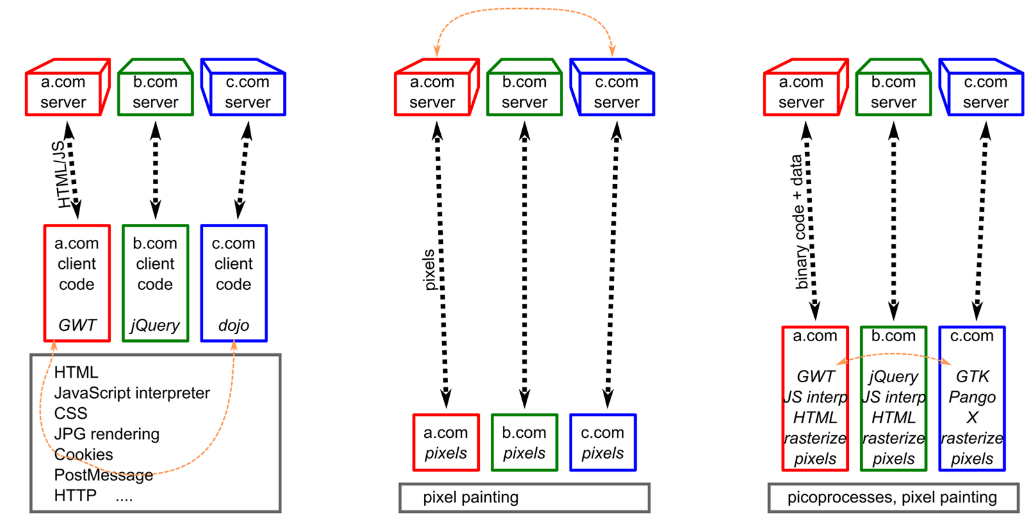
Today a Web browser is a 100 MB operating system. Most of those megabytes interpret JavaScript and render images, but the browser's most important job is to provide the user with the ability to visit different Web sites safely, confident that merely viewing one Web site won't have any effect on any of the other sites she uses and relies on. Reliable isolation is best achieved in a simple design. The ideal Web browser would be a VNC viewer: each site renders its own content entirely independently, and the only job of the client machine is to show the various pixels to the user.

Of course, real browsers don't have such a simple specification. They're vastly more complicated, including HTML, DOM, CSS, JavaScript, JPG, PNG, and a complex specification for how various applications might interact with one another. This complexity forms a vulnerable surface, and hence real Web browsers *don't* actually succeed in isolating different pages; users are cautioned to avoid "dangerous" links lest their browser be compromised.

This ideal VNC pixel browser may seem absurd at first, but clearly it gets isolation right. You might complain that the performance stinks because it depends on a fast, available network, but we can fix that by allowing each site vendor to borrow a little virtual machine on the client; think of it as a *pico-datacenter*. That VM is strongly isolated from the other sites' VMs, just as customers in a real datacenter, say of a cloud-hosting provider, are isolated from one another.

In this new model (Figure 1), the specification of the browser is tiny and robust. Without a simple, clear specification, isolation is unachievable. With a clear specification, like this VM+VNC analogy, seeing how isolation can be rigorously maintained is easy; we push all the challenges of deciding how sites should interact with one another to the sites themselves. Promiscuous sites can still share cookies or engage in risky, CSRF-prone behavior (e.g., hosting user-supplied content), but cautious sites (e.g., bank Web sites) now have the control to reject those complex interactions.

The proposal of a virtual machine for execution and VNC for displaying pixels gives an intuition for how simple the interface can be, but we can go even simpler. We propose a minimal client execution interface called a *picoprocess*. A picoprocess is native code running in a hardware address space. It can allocate memory and threads, use futexes to schedule threads, read a real-time clock, and set a real-time alarm. All communication—to remote servers or to neighboring processes—is via IP; thus, an attacker can't do anything more threatening on the client machine than it could do from a server. (An attacker might relay IP attacks through its presence on a client, but the client's IP packets enter the Internet outside



In today's web, site vendors send code and libraries to the client, but that code runs on a platform of ever-increasing complexity. That platform is responsible for isolating the sites from each other, and it mediates interactions (dashed lines) among the sites.

If the client only supported VNC (painting pixels), its complexity would be drastically reduced, and isolation would be provided with certainty. Display rendering (converting HTML to pixels) happens on the servers, and interactions among sites become protocols among servers. Network performance would be unfortunate, though.

In our design (Embassies), we bring code back to the client, while preserving the simplicity of VNC semantics. Isolation is certain. Interactions among sites are still protocols run by participating vendor code, not mediated by a complex client platform. Because the client accepts native code, complexity grows *inside* apps, not in the platform.

Figure 1: The current, the ideal, and a new way to browse the Web

any firewall, so the relay doesn't gain any privilege from the client's IP address or network position.) The client provides a source of randomness to enable the app to encrypt its messages over IP. Finally, the app displays its content by using its own libraries to render to an off-screen bitmap, then asking the client to paint a rectangle of pixels on the screen, semantics as simple as VNC.

This minimal interface replaces the role of the VM described above. Because it's even simpler than a conventional VM, the interface can be implemented easily on any host, from desktop OSes to native microkernels. On Linux, for example, the picoprocess is a Linux process, blocked from making Linux system calls by one of several mechanisms: kvm, ptrace, or filtering system calls down to `read` and `write` on a single open file handle to a monitor process.

Despite this tiny client picoprocess, the ability to run native code means the app itself can provide glorious complexity. The GIMP photo editor and the AbiWord word processor run in this container [2]. We also run a WebKit browser, to show how the trusted complexity of a conventional HTML browser can be repackaged as safely isolated rendering code.

This idea is ambitious: we're describing a substantial refactoring of the Web, shifting much responsibility from the browser (and the user) to the vendors that create the applications, so that visiting a site is no longer a risky proposition. But the ability to send binary code rather than JavaScript means the idea goes farther: it not only realizes the "safe click" promised by the Web, but it can bring those semantics to classic desktop applications, like the GIMP. When the plan is realized, your Webmail provider might be based on real Outlook and you might edit documents with MicrosoftWord.com or LibreOffice.org: solid desktop app code supported by its site rather than by the end user.

Our Embassies paper [1] proposes this application delivery in detail, discussing the tradeoffs consequent in shifting complexity from clients to applications. Our USENIX paper [2] shows how these complex apps can be repackaged to run inside the constrained picoprocess; source code is available [3].

How Do We Get There from Here?

The overall vision involves reconsidering several of our assumptions about the roles, responsibilities, and relationships that make up today's Web software ecosystem. Rather than end users selecting a JavaScript implementation ("download a fast new browser!"), site vendors will choose their client-side software stacks the same way they choose today among Python, Ruby, and PHP on the server side. Such an ambitious change may need to happen in small steps.

A key step on the way to Utopia is the shift from specifying client-side software in Web 2.0 (the complex amalgamation of JavaScript, DOM, CSS, and so on) to specifying it as native code that interacts through primitive low-level interfaces, such as painting raw pixels. It's an important step because it opens the door to shifting rendering components inside each application.

Figure 2: Three applications that currently run in an Embassies picoprocess

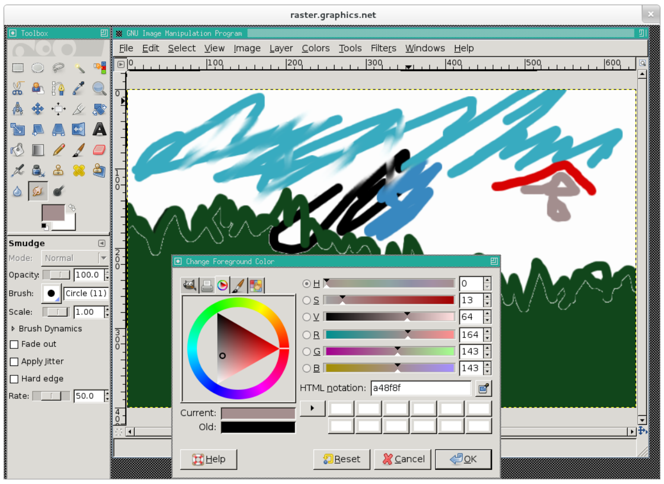
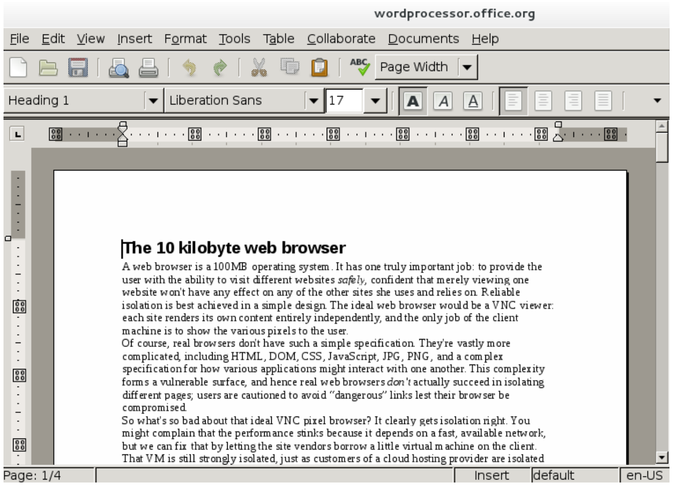


Figure 2a: GIMP in an Embassies picoprocess



The 10 kilobyte web browser

A web browser is a 100MB operating system. It has one truly important job: to provide the user with the ability to visit different websites *safely*, confident that merely viewing one website won't have any effect on any of the other sites she uses and relies on. Reliable isolation is best achieved in a simple design. The ideal web browser would be a VNC viewer: each site renders its own content entirely independently, and the only job of the client machine is to show the various pixels to the user.

Of course, real browsers don't have such a simple specification. They're vastly more complicated, including HTML, DOM, CSS, JavaScript, JPG, PNG, and a complex specification for how various applications might interact with one another. This complexity forms a vulnerable surface, and hence real web browsers *don't* actually succeed in isolating different pages; users are cautioned to avoid "dangerous" links lest their browser be compromised.

So what's so bad about that ideal VNC pixel browser? It clearly gets isolation right. You might complain that the performance stinks because it depends on a fast, available network, but we can fix that by letting the site vendors borrow a little virtual machine on the client. That VM is still strongly isolated, just as customers of a cloud hosting provider are isolated

Figure 2b: AbiWord in an Embassies picoprocess

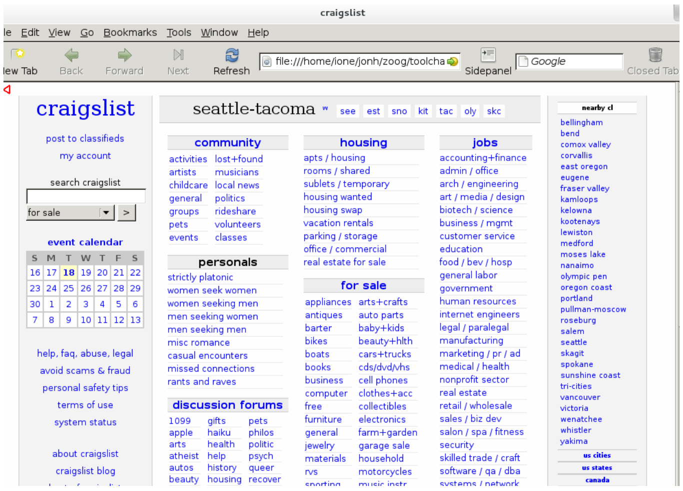


Figure 2c: WebKit in an Embassies picoprocess

But it's also a step that's compelling all by itself. The Xax [4] and Native Client [5] projects, both introduced in 2008, showed that delivering binary code to the client and executing it safely is feasible. Those systems were interesting enough to let us send down interesting components: Doom on NaCl, or PDF and OpenGL renderers on Xax.

Going beyond components to full applications exposes big opportunities. We can already package up GIMP and make it a Web app. We can do the same for the Gnumeric spreadsheet; add a bit of "cloud" and you have made an open-source alternative to Google Docs' spreadsheet. We can fit KDE Marble (a spinning globe) into a picoprocess; that is the foundation of a Google Earth alternative that doesn't require a trusted plugin. The opportunity to deliver rich apps is exciting in itself, even before we reach the ambitious goal of gutting the browser.

Challenges to Delivering Rich Apps

This goal is within reach. We have the technology; however, three tasks remain. First, we need to settle on a suitably shaped native code container. Second, we need to publish a picoprocess browser plugin. Third, we need to wrap up cool apps and publish them as Web apps.

How the Native Code Container Affects Deliverability

We said above that Xax delivered fairly modest stacks of libraries. Xax suffered from a practical burden: a high cost of modifying libraries and applications to run in the new environment. The Xax system replaced the ubiquitous glibc with a patched-together uclibc. In practice, that broke some libraries, and required linking others statically rather than dynamically. This approach worked only for short stacks of libraries. As we tried to enlarge the library stack, each new package required a new effort to disassemble its build system, and some software couldn't even conceive of being built as a static library. These are mundane concerns, but they proved a practical barrier to our ambitions of porting rich desktop apps.

NaCl has encountered similar challenges, for similar reasons. NaCl's isolation mechanism requires modifying the compiler's code generation step to produce code that NaCl can verify is safe. This requirement implies perturbing the build process (and often the link steps) of each package. We suspect that the NaCl team encountered a mundane but tedious and expensive burden much like the one that affected our Xax development.

So the choice of isolation container can have a profound effect on the ease of migrating apps to the new environment. NaCl's choice of verification based on software-fault-isolation (SFI) is driven by a desire to attach untrusted libraries onto the side of an existing browser, right inside the same process. For our ambitions, this objective is a red herring: even today's NaCl libraries don't need tight coupling with the browser; rich apps will stand further alone; and, ultimately, we'd like to see the browser disappear entirely. Because it doesn't offer intra-process isolation, the picoprocess can exploit MMU protection, and hence provides a familiar execution environment for existing code.

Still, that decision wasn't enough to make porting easy in Xax. We made two fine-grained changes from Xax to Embassies that worked out well. The first was that, where Xax allowed the application to control its address space layout, Embassies only allows the app to ask for *how much* memory it needs, not where it goes. This actually increases the burden on the app—the executable must be position-independent—but it makes implementing the host much easier. In Xax, each new host added weird new address-space restrictions; in Embassies, this problem disappeared entirely.

More importantly, the main reason we couldn't use glibc or dynamic libraries in Xax was that we had no support for the x86 segment registers, used for thread-local storage (TLS). That meant we had to compile all components with --no-tls, and we couldn't find a way to use dynamic linking without TLS. The x86 segment-as-TLS is a goofy hack in any case; it uses deprecated hardware to compensate for the architecture's tiny register set. Because contemporary operating systems rely on paging for memory protection, this (ab)use of segmentation hardware has no security risk. By adding it to the Embassies picoprocess x86 specification, we're able to use standard glibc, conventional shared library linkage, and, hence, just about every package as is, with binary compatibility. (This whole discussion is moot on any other, sane architecture, where TLS just uses a conventional program register.)

The result—the Embassies specification for a native code container—is a spec to which a wide variety of rich apps can be ported with little effort. We've ported AbiWord (word processor), Gnumeric (spreadsheet), Gnucash (accounting), Midori (WebKit-based HTML renderer), GIMP (raster graphics design), Inkscape (vector graphics design), and Marble (3D globe). At the same time, the container is small, well-specified and secure, and practical to implement on any host platform.

A Browser Plugin

Now that we know what shape the container should be, achieving the initial step of delivering rich apps as Web apps is within reach: we need to implement the container as a plugin for the popular browsers, and test it for security.

Performance

We've described this new model using a strong analogy to the Web, to appeal to its "safe click" semantics. That doesn't mean we have to keep the Web's online requirement, or that we have to fetch our (now 100 MB) apps every time we open a site.

Whereas conventional browsers include caching behavior, Embassies apps control their own bootstrap and caching. An app can fetch its 100 MB of program image from any cache on the Internet and then check its hash to ensure they are the right bits. That cache can be an untrusted app on the same machine, obviating the need for network connectivity. The local cache can transmit the image in a single IPv6 jumbo frame, making app start fast; we see start time overheads of ~100 ms. Thus "sending big apps" is only an intuitive abstraction borrowed from today's Web; in deployed Embassies, it's fast and works when disconnected.

Once the app is running, native code enables performance better than JavaScript. The picoprocess's isolation comes from paging hardware, and hence introduces no overhead; CPU-intensive GIMP rotations are just as fast inside Embassies as on desktop GIMP.

Delivering Cool Apps

With an appropriate container available as a ubiquitous plugin, it's time to start packaging desktop apps as Web pages. Our ATC '13 paper [2] (and published code [3]) lays out how to achieve this packaging, showing it working for lots of apps, from a spreadsheet to an interactive 3D globe map. These apps need a little modification to make useful Web sites: for example, they need plumbing so that saving a document routes the content to client- or server-side Web storage.

The long-term vision is an exciting one: it promises finally to make browsing "safe," and broadens browsing to include both Web apps and desktop apps. Even if you don't yet buy that vision, the first step down the road is exciting all by itself: delivering all our favorite desktop apps as easily as clicking a link.

References

[1] Jon Howell, Bryan Parno, and John R. Douceur, "Embassies: Radically Refactoring the Web," USENIX Symposium on Networked Systems Design and Implementation (NSDI), awarded "Best Paper," April 2013.

[2] Jon Howell, Bryan Parno, and John R. Douceur, "How to Run POSIX Apps in a Minimal Picoprocess," USENIX Annual Technical Conference (ATC), June 2013.

[3] http://embassies.codeplex.com/.

[4] John R. Douceur, Jeremy Elson, Jon Howell, and Jacob R. Lorch, "Leveraging Legacy Code to Deploy Desktop Applications on the Web," USENIX Symposium on Operating Systems Design and Implementation (OSDI), December 2008.

[5] Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar, "Native Client: A Sandbox for Portable, Untrusted x86 Native Code," IEEE Symposium on Security and Privacy, May 2009.

A Fistful of Bitcoins
Characterizing Payments Among Men with No Names

SARAH MEIKLEJOHN, MARJORI POMAROLE, GRANT JORDAN, KIRILL LEVCHENKO, DAMON MCCOY, GEOFFREY M. VOELKER, AND STEFAN SAVAGE

Sarah Meiklejohn is a PhD candidate in computer science and engineering at the University of California, San Diego. She previously received an ScM in computer science and an ScB in mathematics from Brown University. At UCSD, she is co-advised by Mihir Bellare and Stefan Savage, and has broad research interests in cryptography and security. smeiklej@cs.ucsd.edu

Marjori Pomarole is an undergraduate at the University of California, San Diego, studying computer science. She has interned at Google and Facebook, working with infrastructure monitoring. marjoripomarole@gmail.com

Grant Jordan is a graduate student at the University of California, San Diego, focusing on computer security. His previous work has included research in online spam distribution and botnets, as well as UAV development for the Air Force Research Lab. gejordan@cs.ucsd.edu

Kirill Levchenko is a research scientist at the University of California, San Diego. His research is focused on computer networking and security. klevchen@cs.ucsd.edu

Damon McCoy is an assistant professor in the CS department at George Mason University. He obtained his PhD from the University of Colorado, Boulder, and his research includes work on anonymous communication systems, cyber-physical security, e-crime, and wireless privacy. damon.mccoy@gmail.com

Geoffrey M. Voelker is a Professor of Computer Science at the University of California, San Diego. He works in computer systems, networking, and security. voelker@cs.ucsd.edu

Stefan Savage is a professor of computer science and engineering at the University of California, San Diego. He received his PhD in computer science and engineering from the University of Washington and a BS in applied history from Carnegie Mellon University. Savage is a Sloan Fellow and an ACM Fellow, but is a fairly down-to-earth guy and only writes about himself in the third person when asked. savage@cs.ucsd.edu

Bitcoin is a decentralized virtual currency whose usage has skyrocketed since its introduction in January 2009. Like cash, the ownership of bitcoins is anonymous, as participants transact bitcoins using pseudonyms rather than persistent real-world identities. In this article, we examine the limitations of Bitcoin anonymity and discover that the ability to cluster pseudonyms according to heuristics about shared ownership allows us to identify (i.e., associate with a real-world entity or user) a significant and active slice of the Bitcoin economy. Along the way, we explain a lot about how Bitcoin works.

Bitcoin is a form of electronic cash that was introduced by Satoshi Nakamoto (a pseudonym) in 2008. As its name suggests, Bitcoin is similar to cash in that transactions are irreversible and participants in transactions are not explicitly identified: both the sender(s) and receiver(s) are identified solely by pseudonym, and participants in the system can use many different pseudonyms without incurring any meaningful cost. Bitcoin has two other properties, however, that make it unlike cash: (1) it is completely decentralized, meaning a global peer-to-peer network, rather than a single central entity, acts to regulate and generate bitcoins, and (2) it provides a public transaction ledger, so that although transactions operate between pseudonyms rather than explicit real-world individuals, every such transaction is globally visible.

Since its introduction, Bitcoin has attracted increasing amounts of attention, from both the media and from governments seeking ways to regulate Bitcoin. In large part, much of this attention has been due to either the nature of Bitcoin, which has caused government organizations to express concern that it might enable money laundering or other criminal activity, or to its volatility and ultimate growth as a currency; in late 2012 the exchange rate began an exponential climb, ultimately peaking at $235 US per bitcoin in April 2013, before settling to approximately $100 US per bitcoin (as of September 2013).

In spite of the concerns about Bitcoin, its use of pseudonyms has made gaining any real understanding of how and for what purposes Bitcoin is used a fairly difficult task, as the abstract Bitcoin protocol—if exploited to its fullest extent—provides a fairly robust notion of anonymity. Nevertheless, in modern Bitcoin usage, many users rely on third-party services to store their bitcoins, such as exchanges and wallet services (i.e., banks), rather than individual desktop clients that they operate themselves. In this context, our goal is to exploit this behavior to erode the anonymity of the users that interact with these and other services. In doing so, we do not seek to de-anonymize individual users, but rather to de-anonymize *flows* of bitcoins throughout the network.

Our approach consists of two techniques. First, we engage in a variety of Bitcoin transactions to gain ground-truth data; for example, by depositing bitcoins into an account at the biggest Bitcoin exchange, Mt. Gox, we are able to tag one address as definitively belonging to that service, and by later withdrawing those bitcoins we are able to identify another. To expand on this minimal ground-truth data, we next *cluster* Bitcoin addresses according to two heuristics: one exploits an inherent property of the Bitcoin protocol, and another exploits a current idiom of use in the Bitcoin network. By layering this clustering analysis on top of our ground-truth data collection, we transitively taint entire clusters of addresses as belonging to certain users and services; for example, if our analysis indicated that the address we had previously tagged as belonging to Mt. Gox was contained in a certain cluster, we could confidently tag all of the addresses in that cluster as belonging to Mt. Gox as well.

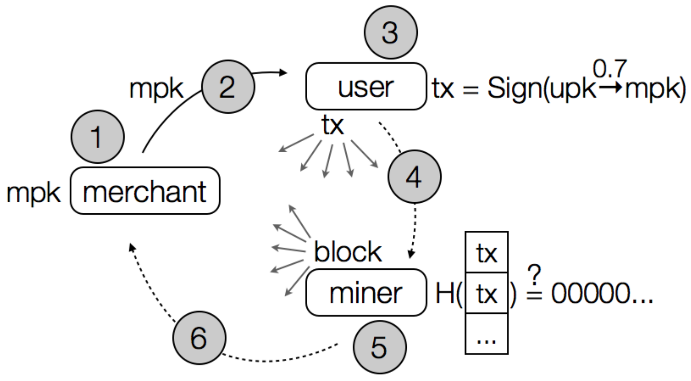

3
mpk
2
user
tx = Sign(upk $\xrightarrow{0.7}$ mpk)
tx
1
4
mpk
merchant
block
tx
miner
H(tx) $\overset{?}{=}$ 00000...
...
6
5

Figure 1: How a Bitcoin transaction works: In this example, a user wants to send 0.7 bitcoins as payment to a merchant. In (1), the merchant generates or picks an existing public key mpk, and (2) sends this public key to the user. By creating a digital signature (3), the user forms the transaction tx to transfer the 0.7 bitcoins from his public key upk to the merchant's address mpk. In (4), the user broadcasts this transaction to his peers, which (if the transaction is valid) allows it to flood the network. In this way, a miner learns about his transaction. In (5), the miner works to incorporate this and other transactions into a block by checking whether their hash is within some target range. In (6), the miner broadcasts this block to her peers, which (if the block is valid) allows it to flood the network. In this way, the merchant learns that the transaction has been accepted into the global block chain, and has thus received the user's payment.

How Bitcoin Works

Before describing our analysis, gaining an understanding of the Bitcoin protocol is necessary. Cryptographically, Bitcoin is composed of two primitives: a digital signature scheme (in practice, ECDSA) and a one-way hash function (in practice, SHA-256). Users' pseudonyms are public keys for the signature scheme, and users can create arbitrarily many pseudonyms by generating signing keypairs. In here and what follows, we use Bitcoin to mean the peer-to-peer network and abstract protocol, and bitcoin, or BTC, to mean the unit of currency; we also use the terms public key, address, and pseudonym interchangeably.

To see how bitcoins get spent, suppose a user has some number of bitcoins stored with one of his pseudonyms. For simplicity, we describe transactions with one input and one output, but transactions can more generally have any number of input and output addresses. To send these bitcoins, the user first creates a message containing (among other things) the intended receiver of the bitcoins, identified by public key, and the transaction in which his pseudonym received the bitcoins. The sender can then sign this message using the private key corresponding to his pseudonym to create a signature. He then broadcasts the signature and message—which together make up the transaction—to his peers, who in turn broadcast it to their peers (see Figure 1).

Before broadcasting the transaction, each peer confirms that the transaction is valid by checking for two things: first, that the signature verifies and thus (by the unforgeability of the signature scheme) was formed correctly by the honest owner of the bitcoins; and second, that no other transaction already used the same previous transaction. This second property is crucial in ensuring that the bitcoins are not double-spent, which is why every peer needs to have access to the entire transaction history (or at least to the transactions in which the received bitcoins have not already been spent). A bitcoin is then not a single object, but rather a chain of these transactions.

After transactions such as these flood the network, they are collected into *blocks*, which serve to timestamp the transactions and further vouch for their validity. The process of creating a block is called *mining*, as it is also the process by which bitcoins are created. Miners (i.e., users seeking to create blocks) first collect all the transactions they hear about into a pool of transactions that have not already been incorporated into blocks; priority often is given to transactions that include a small fee, although at present most transactions do not need to include a fee (the exceptions being transactions that have many inputs and/or outputs, or transactions that carry a large amount of bitcoins). The miner then adds a special *coin generation* transaction to the pool and hashes this collection of transactions.

The miner aims to have a collection of transactions (and other metadata, including a reference to the most recently generated block) that hashes to a value starting with a certain number of zeroes. This and what follows are a somewhat simplified sketch of the mining process; in reality, the miner is trying to generate a hash that is smaller than some target hash. The required number of leading zeroes is proportional to the *difficulty* of the network, which is determined by its current hash rate. The goal is to have the network produce a new block every ten minutes, so the difficulty is adjusted accordingly (e.g., if the hash rate increases, then the difficulty increases as well).

In order to produce this target hash while maintaining the same pool of transactions, the miner also folds in a *nonce* value. One can then think of the mining process as starting with the collection of transactions and the nonce set to 1; if this produces a hash within the target range, then the miner has produced a valid block, and if it doesn't, then she can increment the nonce and try again.

Once the miner does have a valid block, she broadcasts it throughout the network in a manner analogous to the broadcast of transactions, with peers checking the validity of her block by checking whether its hash is within the target range. Her block is accepted into the global transaction ledger after it has been referenced by another block. Because each block references a previous block, blocks form a chain just as transactions do, so this transaction ledger is referred to as the *block chain*.

As a reward for generating this block, which, because of the one-wayness of the hash function, is a computationally intensive task, the miner receives a certain number of bitcoins in the public key specified in her coin generation transaction. This number of bitcoins is determined by the *height* of the block chain: initially, the reward was 50 bitcoins, but at height 210,000 (i.e., after 210,000 blocks were generated, which happened on November 28, 2012), the reward halved, and will continue halving until 21 million bitcoins are generated, at which point the reward will be 0 and miners will be incentivized solely by transaction fees, which will presumably increase as a result.

To summarize, the ledger that every peer downloads when joining the Bitcoin network is the block chain, which consists of a series of blocks, each referencing the one that preceded it. Blocks are accepted into the block chain by consensus: if enough peers agree that a block is valid (for example, it is within the required target range and creates an appropriate number of bitcoins), then they will choose to reference it when generating their own blocks, so that the mining of blocks (and consequent generation of bitcoins) follows a consensus-defined set of rules rather than system requirements. These blocks contain collections of transactions that, like blocks, are validated through their acceptance by peers in the network, which specify the transfer of bitcoins from one set of pseudonyms to another.

Where Bitcoins Are Spent

As of April 13, 2013, the block chain contained more than 16 million transactions between 12 million distinct public keys. More than 11 million bitcoins had been generated (recall that this is more than half of all the bitcoins that will ever be generated), and those bitcoins had been spent many times over, to the point that more than 1 trillion bitcoins had been transacted.

Given this rate of movement, one might naturally wonder where bitcoins are being spent. Since 2010, a variety of Bitcoin services have been introduced at an ever-increasing rate. One of the most widely used categories, *exchanges*, allows users to exchange bitcoins for other currencies, including both fiat currencies such as dollars, and other virtual currencies such as Second Life Lindens. Most of these exchanges also function as banks, meaning they will store your bitcoins for you, although there are also *wallet services* dedicated to doing just that. With all of these services, one runs the risk of theft, which in fact happens fairly often.

Bitcoin mining ASICs were introduced in February 2013 and are capable of computing 64 billion SHA-256 computations per second, meaning the odds of generating a block using just a CPU or even GPU are negligibly small . Due to the computational intensity of generating bitcoins, *mining pools* have become another popular service in the Bitcoin economy, allowing miners to perform some amount of work (e.g., the examination of some slice of the nonce space) and earn fractional bitcoin amounts for every share they contribute.

Users seeking to spend rather than only store or generate bitcoins can do so with a number of merchants, including ones such as WordPress that use the payment gateway BitPay, which accepts payment in bitcoins but pays the merchant in the currency of their choice (thus eliminating all Bitcoin-based risk for the merchant). Users can also gamble with their bitcoins, using poker sites such as BitZino or wildly popular dice games such as Satoshi Dice.

Finally, users seeking to use Bitcoin for criminal purposes can purchase drugs and other contraband on sites such as Silk Road, which are often accessible only via the Tor network. They can also mix (i.e., launder) bitcoins with services such as Bitfog, which promise to take bitcoins and send (to the address of one's choice) new bitcoins that have no association with the ones they received.

The first phase of our analysis involved interacting with these and many other services. In total, we kept accounts with 26 exchanges and ten wallet services, and made purchases with 25 different vendors, nine of which used the payment gateway BitPay; a full list of the services with which we interacted can be found in Table 1, and images of our tangible purchases can be found in Figure 2. We engaged in 344 transactions

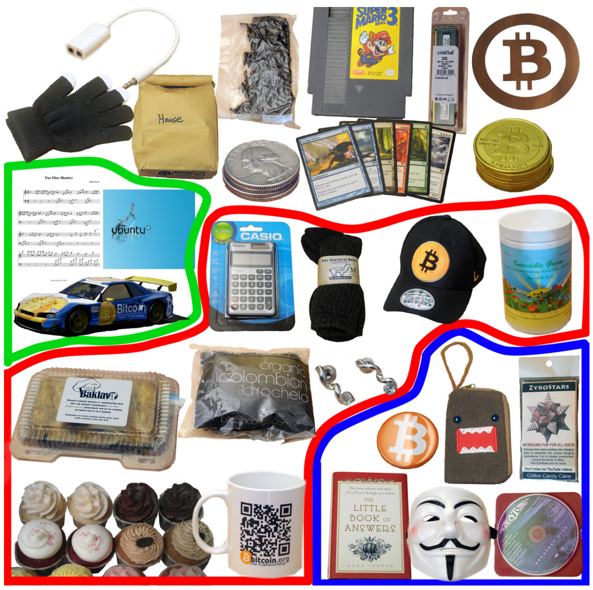


Figure 2: The physical items we purchased with bitcoins, ranging from beef jerky from BitPantry to a used Boston CD from Bitmit. The items in green were purchased from CoinDL (the "iTunes of Bitcoin"), in blue from Bitmit (the "eBay of Bitcoin"), and in red using the payment gateway BitPay.

with these services, which allowed us definitively to tag 832 addresses (recall that transactions can have arbitrarily many input addresses, which allows us to tag multiple addresses per transaction). We additionally scraped various publicly claimed addresses that we found, such as users' signatures in Bitcoin forums, although we were careful to use only tags for which we could perform some manual due diligence.

Clustering Bitcoin Addresses

In theory, the use of pseudonyms within Bitcoin provides a property called *unlinkability*, which says that users' transactions using one set of pseudonyms should not be linked to their transactions using a different set of pseudonyms. In practice, however, certain properties of Bitcoin usage erodes this anonymity.

Recall that, in order to create a valid Bitcoin transaction, the sender must know the private signing key corresponding to the public key in which the bitcoins are held. Now suppose that a user wishes to send 10 BTC to a merchant, but has 4 BTC in one address and 6 BTC in another. One potential way to pay the merchant would be to create a new address, send the 4 BTC and 6 BTC to this new address, and then send the 10 BTC now contained in this new address to the merchant. (In fact, this is the method that preserves the most anonymity.) Instead, the Bitcoin protocol allows for a simpler and more efficient solution: transactions can have arbitrarily many inputs, so the 4 BTC and 6 BTC addresses can be used as input to the same transaction, in which the receiver is the merchant.

Mining			
	50 BTC	BTC Guild	Itzod
	ABC Pool	Deepbit	Ozcoin
	Bitlockers	Eclipse MC	Slush
	Bitminter	Eligius	
Wallets			
	Bitcoin Faucet	Easywallet	Strongcoin
	My Wallet	Flexcoin	WalletBit
	Coinbase	Instawallet	
	Easycoin	Paytunia	
Exchanges			
	Bitcoin 24	BTC-e	Aurum Xchange
	Bitcoin Central	CampBX	BitInstant
	Bitcoin.de	CA VirtEx	Bitcoin Nordic
	Bitcurex	ICBit	BTC Quick
	Bitfloor	MercadoBitcoin	FastCash4Bitcoins
	Bitmarket	Mt Gox	Lilion Transfer
	Bitme	The Rock	Nanaimo Gold
	Bitstamp	Vircurex	OKPay
	BTC China	Virwox	
Vendors			
	ABU Games	BTC Buy	HealthRX
	Bitbrew	BTC Gadgets	JJ Games
	Bitdomain	Casascius	NZBs R Us
	Bitmit	Coinabul	Silk Road
	Bitpay	CoinDL	WalletBit
	Bit Usenet	Etsy	Yoku
	Bit Elfin	BitZino	Gold Game Land
	Bitcoin 24/7	BTC Griffin	Satoshi Dice
	Bitcoin Darts	BTC Lucky	Seals with Clubs
	Bitcoin Kamikaze	MTC on Tilt	
	Bitcoin Minefield	Clone Dice	
Miscellaneous			
	Bit Visitor	Bitfog	CoinAd
	Bitcoin Advertisers	Bitlaundry	Coinapult
	Bitcoin Laundry	BitMix	Wikileaks

Table 1: We interacted with many services, and provide approximate groupings as shown here.

This observation gives rise to our first clustering heuristic: if two addresses have been used as input to the same transaction, they are controlled by the same user. This heuristic is quite safe, as the sender must know the private keys corresponding to all input addresses in order to form a valid transaction, and as such it has already been used in the Bitcoin literature to the point where freely available tools exist online for performing this analysis.

Our second clustering heuristic expands on this first heuristic and exploits the way in which change is made. In the Bitcoin protocol, when an address receives some number of bitcoins, it has no choice but to spend those bitcoins all at once (recall that this is because each transaction must reference a previous transaction, and transactions cannot be referenced multiple times). If this number of bitcoins is in excess of what the sender wants to spend (e.g., if he has 4 BTC stored in an address and wants to send 3 BTC to a merchant), then he creates a transaction with two outputs: one for the actual recipient (e.g., the merchant receiving 3 BTC) and one change address that he controls and can use to receive the change (e.g., the 1 BTC left over).

This behavior gives rise to our second clustering heuristic: the change address in a transaction is controlled by the sender. As change addresses do not a priori look any different from other addresses, significant care must be taken in identifying them. As a first step, we observe that in the standard Bitcoin client, a change address is created internally and is not even known to the user (although he can always learn it by examining the block chain manually). Furthermore, these change addresses are used only twice: once to receive the change in a transaction, and once to spend their contents fully as the input in another transaction (in which the client will create a fresh address to receive any change).

By examining transactions and identifying the outputs that meet this pattern of one-time usage, we identify the change addresses. If more than one output meets this pattern, then we err on the side of safety and do not tag anything as a change address. Using this pattern—with a number of additional precautions, such as waiting a week to identify change addresses—we identified 3.5 million change addresses, with an estimated false positive rate of 0.17%, noting that the false positive rate can only be estimated, as in the absence of ground-truth data we cannot know what truly is and isn't a change address. By then clustering addresses according to this heuristic, we collapsed the 12 million public keys into 3.3 million clusters.

Putting It All Together

By layering our clustering analysis on top of our ground-truth data (and thus transitively tagging entire clusters that contain previously tagged addresses), we were able to identify 1.9 million public keys with some real-world service or identity, although in many cases the identity was not a real name, but rather (for example) a username on a forum. Although this is a somewhat small fraction (about 16%) of all public keys, it nevertheless allows us to de-anonymize significant flows of bitcoins throughout the network.

Toward this goal, we first examined interactions with known Bitcoin services. By identifying a large number of addresses for various services (e.g., we identified 500,000 addresses as controlled by Mt. Gox, and more than 250,000 addresses as controlled by Silk Road), we were able to observe interactions with these services, such as deposits into and withdrawals from exchanges. Although this does not de-anonymize the individual participating in the transaction (i.e., we could see that a user was interacting with a service, but did not necessarily know which user), it does serve to de-anonymize the flow of bitcoins into and out of the service.

To demonstrate the usefulness of this type of analysis, we turned our attention to criminal activity. In the Bitcoin economy, criminal activity can appear in a number of forms, such as dealing drugs on Silk Road or simply stealing someone else's bitcoins. We followed the flow of bitcoins out of Silk Road (in particular, from one notorious address) and from a number of highly publicized thefts to see whether we could track the bitcoins to known services. Although some of the thieves attempted to use sophisticated mixing techniques (or possibly mix services) to obscure the flow of bitcoins, for the most part tracking the bitcoins was quite straightforward, and we ultimately saw large quantities of bitcoins flow to a variety of exchanges directly from the point of theft (or the withdrawal from Silk Road).

As acknowledged above, following stolen bitcoins to the point at which they are deposited into an exchange does not in itself identify the thief; however, it does enable further de-anonymization in the case in which certain agencies can determine (through, for example, subpoena power) the real-world owner of the account into which the stolen bitcoins were deposited. Because such exchanges seem to serve as chokepoints into and out of the Bitcoin economy (i.e., there are few alternative ways to cash out), we conclude that using Bitcoin for money laundering or other illicit purposes does not (at least at present) seem to be particularly attractive.

Dowser: A Guided Fuzzer for Finding Buffer Overflow Vulnerabilities

ISTVAN HALLER, ASIA SLOWINSKA, MATTHIAS NEUGSCHWANDTNER, AND HERBERT BOS

Istvan Haller is a PhD student in the Systems and Network Security group at the Vrije Universiteit Amsterdam. His current research focuses on automatic analysis of software systems and its application to enhance system security. i.haller@vu.nl

Asia Slowinska is an assistant professor in the Systems and Network Security group at the Vrije Universiteit Amsterdam. Her current research focuses on developing techniques to automatically analyze and reverse engineer complex software that is available only in binary form. asia@few.vu.nl

Matthias Neugschwandtner is a PhD student at the Secure Systems Lab at the Vienna University of Technology. mneug@iseclab.org

Herbert Bos is a full professor in Systems and Network Security at Vrije Universiteit Amsterdam. He obtained his PhD from Cambridge University Computer Laboratory (UK). He is proud of all his (former) students, three of whom have won the Roger Needham PhD Award for best PhD thesis in systems in Europe. In 2010, Herbert was awarded an ERC Starting Grant for a project on reverse engineering that is currently keeping him busy. herbertb@few.vu.nl

Buffer overflows have long plagued existing software systems, making them vulnerable to attackers. Our tool, Dowser, aims to tackle this issue using efficient and scalable software testing. Dowser builds on a new software testing paradigm, which we call dowsing, that focuses the testing effort around relevant application components. This paradigm proved successful in practice, as Dowser found real bugs in complex applications such as the nginx Web server and the ffmpeg multimedia framework.

Buffer overflows represent a long-standing problem in computer science, first identified in a US Air Force study in 1972 [2] and famously used in the Morris worm in 1988. Even after four decades, buffer overflows are perennially in the top three of the most dangerous software errors. Recent studies [8] suggest that the situation will not change soon. One way to handle them is to harden the binary using stack canaries, address space randomization, and the like in the hope that the program will crash when the buffer overflow bug is triggered; however, although crashing is better than being pwned, crashes are undesirable, too.

Thus, vendors prefer to squash bugs beforehand and typically try to find as many as they can by means of fuzz testing. Fuzzers feed programs invalid, unexpected, or random data to see whether they crash or exhibit unexpected behavior. Recent research in testing has led to the development of whitebox fuzzing [3, 4, 5]. By means of symbolic execution, whitebox fuzzing exercises all possible execution paths through the program and thus uncovers all possible bugs, although it may take years to do.

Imagine that you are a software tester and you are given a binary, without knowledge about the application internals or its specification. Where do you start? What features will you be looking for? Intuitively, you start from some random input that you refine based on the observed output. Seeing that you will spend most of your time figuring out the input semantics, instead of testing the underlying functionality itself, is not difficult. These are the same challenges that symbolic execution faces when testing applications without developer input.

In this article, we introduce an alternative testing approach that we call dowsing. Rather than testing all possible execution paths, this technique actively searches for a given family of bugs. The key insight is that careful analysis of a program lets us pinpoint the right places to probe and the appropriate inputs to do so. The main contribution is that our fuzzer directly homes in on the bug candidates and uses a novel "spot-check" approach in symbolic execution. Specifically, Dowser applies this approach to buffer overflow bugs, where we achieve significant speed-ups for bugs that would be hard to find with most existing symbolic execution engines.

In summary, Dowser is a new fuzzer targeted at vendors who want to test their code for buffer overflows and underflows. We implemented the analyses of Dowser as LLVM [7] passes, whereas the symbolic execution step employs S2E [4]. Finally, Dowser is a practical solution. Rather than aiming for all possible security bugs, it specifically targets the class of buffer overflows (one of the most, if not *the* most, important class of attack vectors for code injection). So far, Dowser has found several real bugs in complex programs such as nginx, ffmpeg, and inspircd. Most of them are extremely difficult to find with existing symbolic execution tools.

Dowsing for Candidate Instructions

Dowser builds on the concept of vulnerability candidates, that is, program locations that are relevant to a specific bug type, in our case buffer overflows. In other words, it scans the binary for features that are possible indications for those hard-to-find buffer overflows. For instance, for a buffer overflow to occur, we need code that accesses buffers in a loop. Additionally, we build on the intuition that code with convoluted pointer arithmetic and/or complex control flow is more prone to such memory errors than straightforward array accesses. Moreover, by focusing on such code, Dowser prioritizes bugs that are complicated—typically, the kind of vulnerabilities that static analysis or random fuzzing cannot find. The aim is to reduce the time wasted on shallow bugs that could also have been found using existing methods. In this section, we explain how we identify and rank the vulnerability candidates.

Identifying the Interesting Spots

Previous research has shown that complex code really is more error prone than simple code for bugs in general; however, Zimmermann et al. [9] also argued that we need metrics that exploit the unique characteristics of specific vulnerabilities, e.g., buffer overflows or integer overruns. So how do we design a good metric for buffer overflows?

Intuitively, convoluted pointer computations and control flows are hard to follow by a programmer, and thus more bug prone. Therefore, we select vulnerability candidates, by focusing on “complex” array accesses inside loops. Further, we limit the analysis to pointers that evolve together with loop induction variables, the pointers that are repeatedly updated to access (various) elements of an array.

Prioritize, Prioritize, Prioritize...

After obtaining a set of vulnerability candidates, Dowser prioritizes them according to the 80-20 Pareto principle: we want to discover the majority of software bugs while testing only a subset of the potentially vulnerable code fragments. While all the array accesses that evolve with induction variables are potential targets, Dowser prioritizes them according to the complexity of the data- and control-flows for the array index (pointer) calculations.

For each candidate loop, it first statically determines (1) the set of all instructions involved in modifying an array pointer (we will call this a pointer’s analysis group), and (2) the conditions that guard this analysis group (for example, the condition of an if or while statement containing the array index calculations). Next, it labels all such sets with scores reflecting their complexity. It may happen that the data-flow associated with an array pointer is simple, but the value of the pointer is hard to follow due

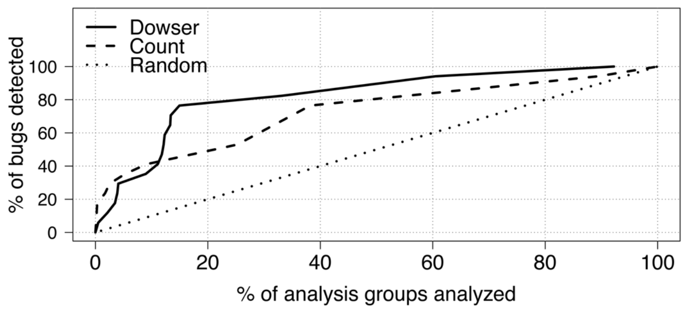


Figure 1: A comparison of random testing and two scoring functions: Dowser’s and count. It illustrates how many bugs we detect if we test a particular fraction of the analysis groups.

to some complex control changes. For this reason, Dowser also considers the complexity of the variables involved in conditionals. For a detailed description of the procedure, refer to [6].

We emphasize that our complexity metric is not the only way to rank the buffer accesses. For instance, we could also use the length of a loop, the number of computations involved in the computation of the array index, or some other heuristic. In fact, Dowser does not care which ranking function is used, as long as it prioritizes the accesses in the best possible way. In our lab, we have evaluated several such functions and, so far, the complexity metric performed best. For instance, Figure 1 compares Dowser’s complexity metric to count, a straightforward scoring function that simply counts the number of instructions involved in the computation of the array pointer.

We base the evaluation on a set of known vulnerabilities from six real world programs: nginx, ffmpeg, inspircd, libexif, poppler, and snort. Additionally, we consider the vulnerabilities in sendmail tested by Zitser et al. [10]. For these applications, we analyzed all buffer overflows reported in CVE since 2009 to find 17 that match our vulnerability model. Figure 1 illustrates the results. Random ranking serves as a baseline; clearly both count and Dowser perform better. In order to detect all 17 bugs, Dowser must analyze 92.2% of all the analysis groups; however, even with only 15% of the targets, we find almost 80% (13/17) of all the bugs. At that same fraction of targets, count finds a little more than 40% of the bugs (7/17). Overall, Dowser outperforms count beyond the 10% in the ranking, and it reaches the 100% bug score earlier than the alternatives, although the difference is minimal.

Efficient Spot-Checking

The main purpose of spot-checking is to avoid the complexity stemming from whole-program testing. For example, the nginx-0.6.32 Web server [1] contains a buffer underrun vulnerability, where a specially crafted input URI tricks the program into setting a pointer to a location outside its buffer boundaries. When this pointer is later used to access memory, it allows attackers to overwrite a function pointer and execute arbitrary code on the system. Exhaustively testing the Web server to find this bug is almost impossible due to the complexity of the HTTP packets used as input. Indeed, the existing tools didn't discover the vulnerability within eight hours. Dowser, however, ranked the vulnerable array access at the fourth most complex out of a total of 62 potentially vulnerable loops, and then found the bug within five minutes.

As a baseline, spot-checking uses concolic execution [5], a combination of concrete and symbolic execution, where the concrete (fixed) input starts off the symbolic execution. Dowser enhances concolic execution with the following two optimizations.

Finding Relevant Inputs

Typically only a part of the input influences a particular analysis group. In our example, only the URI field from the HTTP packet is relevant to the faulty parser. Dowser aims to identify and enforce this correlation automatically. In technical terms, Dowser uses dynamic taint analysis to determine which input fields influence pointers dereferenced in the analysis group. During the testing phase, Dowser only treats those fields as symbolic and keeps the remaining ones unchanged.

Eliminating Irrelevant Code

The second optimization leverages the observation that only the program instructions that influence the underlying pointer arithmetic are relevant to buffer overflows. Thus, when checking a particular spot, that is, a buffer access, Dowser analyzes the associated loop a priori to find branch outcomes that are most likely to lead to new pointer values. The results of this analysis are used to focus the testing effort around the most relevant program paths. In the URI parser example, it would prioritize branches that impact pointer arithmetic, and ignore those that only affect the parsing result.

Dowser's loop exploration procedure operates in two main phases: learning and bug finding. In the learning phase, Dowser assigns each branch a weight approximating the probability that a path following this direction contains new pointer dereferences. The weights are based on statistics of pointer value variance observed during symbolic execution with limited inputs.

In the bug finding phase, Dowser symbolically executes a real-world-sized input in the hope of finding inputs that trigger a bug. Dowser uses the weights from the learning phase to steer its symbolic execution toward new and interesting pointer dereferences. The goal of our heuristic is to avoid execution paths that are redundant from the point of view of pointer manipulation. Thus, Dowser shifts the target of symbolic execution from traditional code coverage to pointer value coverage. Therein lies the name we gave to this new search heuristic, Value Coverage Search, to emphasize the data-centric approach that Dowser takes.

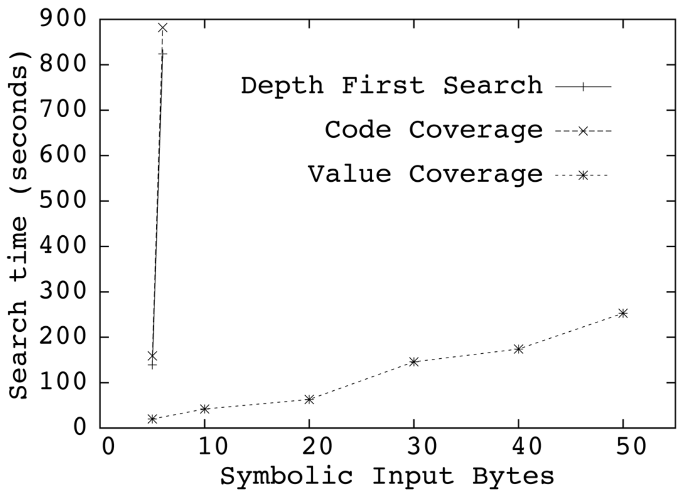


Figure 2: A comparison of the different search heuristics while testing for the vulnerability in nginx. In all instances the symbolic input is limited to the URI field.

We highlight the benefits gained via spot-checking on the same nginx example used so far. As mentioned in the beginning of this section, the application itself is too complex for the baseline concolic execution engine, which was unable to trigger the bug within eight hours. Limiting the symbolic input to the given URI field does allow S2E to detect the bug using its built-in search heuristics (Depth-First Search and Code Coverage), as we show in Figure 2; however, the reader can also notice an exponential explosion in the search time, making the traditional search heuristics inefficient when the input size grows beyond six bytes. Although many tools recommend code coverage [5] as the primary strategy to find bugs, in our experience it does not help with buffer overflows, because memory corruptions require a particular execution context. Even if 100% code coverage is reached, these bugs may stay undetected. In contrast with these results, our Value Coverage heuristic shows excellent scalability with an almost linear increase in execution time in relation with the input size.

Dowser in the Real World

Dowser detected nine memory corruptions from six real-world applications of several tens of thousands LOC, including the ffmpeg videoplayer of300k LOC. The other applications that we looked at were nginx, inspircd, poppler, libexif, and snort. The bug in ffmpeg and one of the bugs in poppler were also not documented before. We run S2E for as short a time as possible, (e.g., a single request/response in nginx and transcoding a single frame in ffmpeg). Still, in most applications, vanilla S2E fails to find bugs within eight hours, whereas Dowser is always capable of triggering the bug within 15 minutes of testing the appropriate analysis group. More details about the evaluation can be found in our paper [6].

Although our paper applies dowsing to the concrete class of buffer overflows, the underlying principles are also valid for a wide variety of bug families. Once we identify the unique feature set characterizing each of them, we will be able to discover more vulnerable locations. Recent developments in the analysis of legacy binaries also suggest that the techniques required by Dowser may soon be applicable without the need of source code information. Such developments would enable the efficient testing of legacy binaries to learn about possible zero-day attacks within.

Conclusion

Dowser is a guided fuzzer that combines static analysis, dynamic taint analysis, and symbolic execution to find buffer overflow vulnerabilities deep in a program's logic. It leverages a new testing approach, called dowsing, that aims to actively search for bugs in specific code fragments without having to deal with the complexity of the whole binary. Dowser is a new, practical, and complete fuzzing approach that scales to real applications and complex bugs that would be hard or impossible to find with existing techniques.

Acknowledgments
This work is supported by the European Research Council through project ERC-2010-StG 259108-ROSETTA, the EU FP7 SysSec Network of Excellence and by the Microsoft Research PhD Scholarship Program through the project MRL 2011-049. The authors would like to thank Bartek Knapik for his help in designing the statistical evaluation.

References

[1] CVE-2009-2629: Buffer Underflow Vulnerability in Nginx: http://Cve.Mitre.Org/Cgi-Bin/Cvename.Cgi?Name =CVE-2009-2629, 2009.

[2] J. P. Anderson, "Computer Security Technology Planning Study," Tech. Rep., Deputy for Command and Management System, USA, 1972.

[3] C. Cadar, D. Dunbar, and D. Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs," Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (2008), OSDI '08.

[4] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: A Platform for In Vivo Multi-Path Analysis of Software Systems," Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (2011), ASPLOS '11.

[5] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated Whitebox Fuzz Testing," Proceedings of the 15th Annual Network and Distributed System Security Symposium (2008), NDSS '08.

[6] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos, "Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations," Proceedings of USENIX Security '13 (Washington, DC, August 2013), USENIX.

[7] C. Lattner and V. Adve, "Llvm: A Compilation Framework for Lifelong Program Analysis and Transformation," Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO '04) (Palo Alto, California, March 2004).

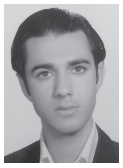
[8] V. van der Veen, N. Dutt-Sharma, L. Cavallaro, and H. Bos, "Memory Errors: The Past, the Present, and the Future," Proceedings of the 15th international Symposium on Research in Attacks, Intrusions and Defenses (2012), RAID '12.

[9] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista," Proceedings of the 3rd international Conference on Software Testing, Verification and Validation (April 2010), ICST '10.

[10] M. Zitser, R. Lippmann, and T. Leek, "Testing Static Analysis Tools Using Exploitable Buffer Overflows from Open Source Code," Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering (2004), SIGSOFT '04/FSE-12.

Rent to Pwn
Analyzing Commodity Booter DDoS Services

MOHAMMAD KARAMI AND DAMON MCCOY


Mohammad Karami is a second year PhD student in information security and assurance at George Mason University. His broad research interests include security, trust, and privacy issues in open distributed systems. More recently, he has been working on studying and measuring malicious behavior of financially motivated underground organizations. mkarami@masonlive.gmu.edu

Damon McCoy is an assistant professor in the CS department at George Mason University. Previously he was a Computer Innovation Fellow at the University of California, San Diego. He obtained his PhD from the University of Colorado, Boulder, and his research includes work on anonymous communication systems, cyber-physical security, e-crime, and wireless privacy. mccoy@cs.gmu.edu

Distributed denial-of-service (DDoS) attacks, the practice by which a malicious party attempts to disrupt a host or network service, has become an increasingly common and effective method of attack. In this article, we summarize what we have learned while investigating the phenomenon of what are called booter or stresser services. These booter services began as a tool used by video-game players to gain an advantage by slowing or disrupting their opponents' network connection for a short period of time; however, as these services have become increasingly commercialized, they have morphed into powerful, reliable, and easy to use general purpose DDoS services that can be linked to several attacks against non-gamer Web sites.

We begin with an overview of DDoS techniques. We then outline the common capabilities and infrastructure used by these booter services supported with information found on underground forums that market and review such services. Finally, we present empirical measurements of one particular booter, known as TwBooter, based on a publicly leaked dump of their operational database and our own measurements of their attack capabilities.

Background on DDoS Attack Methods

Well honed DDoS methods can amplify the amount of traffic an attacker is able to generate by an order of magnitude. Also, there are many attacks that take advantage of misconfigured options present in many Web servers to magnify the effectiveness of an attack. Although booter services are not as technologically advanced as cutting-edge DDoS malware, such as Dirt Jumper Drive [3], they implement several of the most effective DDoS attacks. We review a few of the methods that are implemented by most booter services in order to provide an idea of their sophistication.

SYN flood. This form of DoS attack is conducted by rapidly sending large numbers of TCP SYN requests. To make these requests difficult to filter, the IP source address is normally spoofed. The goal of this attack is to force a server to expend a large amount of resources handling these requests, so that it does not have enough resources left to respond to legitimate requests.

DNS reflection. This method enables an attacker to consume all of the victim's bandwidth by amplifying their traffic by a factor of ten or more times the amount of actual traffic the attacker is able to send. The attack takes advantage of several facts. The first is that well-crafted DNS requests can produce DNS replies that are more than ten times larger. The next is that DNS operates over UDP, which is a connectionless protocol; thus the attacker can send a spoofed DNS request that causes the large DNS reply message to be directed to the victim. The last key part of this attack is that there are large numbers of what are called "open DNS resolvers." These are misconfigured DNS resolvers that will provide resolution for clients outside its administrative domain.

HTTP GET/HEAD/POST flood. This attack focuses directly on the Web servers and operates by making a large number of HTTP requests to the Web server, with the goal of triggering database queries or other processes that consume large amounts of server resources.

RUDY/Slowloris. RUDY stands for "aRe yoU Dead Yet," and it again targets Web servers, specifically HTTP forms, with long POST arguments that cause vulnerable servers to exhaust their pool of connections processing these never-ending HTTP POST requests. Another twist on this attack is slowloris, which slowly reads HTTP replies to tie up and exhaust the available pool of connections.

The Underground View of Booter Services

Booter services are relatively easy to locate, and there are countless numbers of them in operation as of the writing of this article. They can be found by Web searches for "booter stresser," and they publicly market themselves as network stress testing services in order to maintain a facade of legitimacy; however, on underground forums, such as hackforums.net, they market themselves as DDoS services that "hit hard" and offer a number of add-on services, such as locating a victim's IP via their Skype ID and a server's real IP address to get around CloudFlare and other anti-DDoS services.

Most of these booter services operate on a subscription model, in which their customers pay a monthly fee that enables them to launch as many DDoS attacks as they want for the month. A basic membership costs around $10–$30 US per month and normally entitles the customer to only one concurrent attack that lasts 30–60 seconds. The subscriber can launch unlimited new attacks after their current one has ended. In order to launch more than one concurrent attack or attacks that last longer (from one to three hours) the customer must purchase more costly premium subscriptions that range in cost from $50–$200 US per month. Most booter services accept payment via PayPal and some accept bitcoins.

On these same underground forums there are advertisements from hosting ISPs that rent servers and are tolerant of launching DDoS attacks. These advertisements and comments from the operators of these booter services indicate that many of them are renting dedicated servers instead of using compromised servers or large botnets for their attack infrastructure. Determining whether a server is rented by an attacker or compromised is difficult; however, from a business perspective, renting servers might make sense because rented servers are likely more stable than compromised servers or botnets.

Additionally, we see many posts on these underground forums from booter service operators claiming they have updated their lists of open DNS resolvers and proxy lists. This provides anecdotal evidence they are exploiting other organizations' misconfigured DNS resolvers for DNS reflection attacks and using public proxies to make it more difficult to filter Web server attacks launched from a small set of dedicated servers via IP address.

Finally, there are posts that indicate many of these booter services are based on code that has leaked or been stolen, such as the asylum booter source code, available at its Web site [1]. This reinforces the fact that there is a low barrier of entry for starting a booter service.

An Analysis of the TwBooter Service

To gain a deeper understanding of booter services, we conducted an empirical analysis of TwBooter (http://booter.tw). We will present analysis based on various aspects of TwBooter's operations, including the infrastructure leveraged for mounting DDoS attacks, details on service subscribers, and the targets being victimized by the booter. Although TwBooter isn't thought to be among the largest booter services, it recently has attracted attention after being linked to a series of DDoS attacks targeting a popular blog on computer security and cybercrime [5] and the Ars Technica Web site [2].

Data Set

Most of our analysis is based on a publicly available SQL dump file of the operational database of the TwBooter service. The data set covers a period of 52 days ending on March 15, 2013, and contains more than 48,000 attack records. Table 1 provides a summary of the data contained in this data set. See our paper [4] for more details on what this data set included.

Duration	Clients	Victims	Attacks
Jan. 2013–Mar. 2013	312	11174	48844

Table 1: Summary of TwBooter data set used in the analysis

Ethics, Legal, Authenticity Implications

When dealing with a leaked data set, many issues must be addressed before using it. Two of the key issues when dealing with potentially stolen data is that the data is used in an ethical and legal fashion. In this case, the data was publicly leaked and previously reported upon, and so we designed a methodology that would minimize any additional harm from our analysis and publication. Specifically, we omitted personal information from our publication, such as email addresses and names of the subscribers, victims (except in the cases were the information was publicly reported), and operators of this service even when these details were known. Another key issue when dealing with data of unknown provenance is checking as much as possible that it is authentic and accurate. For this data set, we contacted three of the victims and confirmed that the data correlated with attacks that they experienced. We also checked to make sure the data was internally consistent. This gives us some confidence that this data is not completely fabricated; however, some of the data could be fabricated or inaccurate.

Attack Infrastructure

Our analysis of the TwBooter leaked data indicates that only 15 distinct servers were used to perform all the attacks launched by this service. This means that TwBooter relies on a smaller set of servers to perform DDoS attacks. Compared to clients, servers utilized for this purpose could be much more effective as they typically have higher computational and bandwidth capacities, making them more capable of starving bandwidth or other resources of a targeted system.

Further analysis shows that only three servers have been active for the entire 52-day period covered by our data. The other servers either left or joined the pool of servers in the middle of the period. A total of nine servers were in active operation as of March 15. The lifetime for the six inactive servers ranged from three days to 16 days, with an average of 11 days. The average lifetime of nine servers that were still active was 32 days. Two of the servers were hosted in the USA and the rest were hosted by an ISP located in the Netherlands. We omit the name of the ISPs because we do not have enough evidence to tell whether the servers have been compromised or have been directly leased from the hosting providers. This supports the anecdotal evidence that booter services have a relatively stable attack infrastructure based on higher powered servers.

Attack Measurement

Although TwBooter implemented 12 different attack types, the ones mentioned above account for more than 96% of all performed attacks. To measure the effectiveness of these attacks, we subscribed to TwBooter and initiated a number of attacks to one of our own servers. Table 2 summarizes the measurement results for both a SYN flood and UDP flood. The UDP flood used a DNS reflection and amplification attack to generate 827 Mb/sec of DNS query response traffic directed at our server by sending out large numbers of forged DNS request queries that included our server's IP address as the IP source address. For the SYN flood, we observed 93,750 TCP SYN requests per second with randomly spoofed IP addresses and port numbers directed at our server.

In addition to these two flood attacks, we also launched both HTTP GET/POST attacks on our server to see whether proxy servers were utilized by TwBooter. We observed a total of 26,296 distinct proxy servers being used for a five-minute HTTP GET attack and 21,766 proxy servers for an HTTP POST attack of the same length.

Attack type	# of packets	Avg. packet size	Volume
UDP flood	4552899	1,363 bytes	827 Mb/sec
SYN flood	5625086	54 bytes	40 Mb/sec

Table 2: Summary of measured attacks (duration 60 secs)

Customers

A total of 277 active users subscribed to the TwBooter within the time period of the data set. The subscription information and information on the cost of each combination of options allows us to estimate that TwBooter earned $7,727 a month. Assuming they were paying around $250–$300/month each for nine dedicated servers at a hosting ISP, this would be a profitable enterprise.

To make our analysis easier to understand, we classified users into three categories of behavior based on their subscription type: (1) gamers mounting short-lived attacks of no longer than 10 minutes, (2) Web site attackers with attacks lasting between one and two hours, and (3) privileged users with the right to initiate attacks lasting for more than two hours. Some users could not be easily categorized into one of these groups and were excluded from the analysis. The users assigned to one of the three groups account for about 83% of all users.

The intuition behind this method of classification is that TwBooter utilizes high bandwidth servers to mount DDoS attacks. Gamers typically use residential Internet connections to play online games. Considering the limited capacity of a gamers' links, they can be easily overwhelmed with large amounts of traffic originated from one server for a short period of time. For this reason, the majority of TwBooter users targeting gamers have subscribed for short-lived DDoS attacks. We found that users who subscribed for durations of between 10 minutes to less than an hour were difficult to classify, and thus we have left them out of this analysis. Those subscribed for an attack duration of an hour or more are likely to be users targeting Web sites. Interestingly, there are a few users who have the privilege to initiate attacks lasting more than two hours, an option that is not available to ordinary users at registration time.

Table 3 summarizes service usage for the three groups of users. As observed, gamers and Web site attackers exhibit similar behavior in terms of the average number of attacks initiated per day and the number of distinct victims targeted per day. Users in the third group, however, behave differently. Although privileged users tend to target fewer distinct victims per day, they initiate more attack instances on those targets. This is probably attributable to the fact that the privileged users are more likely to utilize concurrent attacks.

	Gamers	Web site	Privileged
Number of users	180	41	8
Avg. distinct targets per day	3.32	3.46	2.86
Avg. attacks per day	13	13	16
Avg. attack time per day	59 m	14 h	105 h

Table 3: Service usage of the three user groups

In terms of the average number of attacks initiated per day, we observe that users in all of the three groups use the service fairly heavily. As expected, the average amount of time spent having an attack carried out varies significantly among each of the user groups. Although the maximum duration of an attack for gamers and Web site attackers is ten minutes and two hours, respectively, we have attack records for privileged users that last for a few days. Besides the privilege of mounting longer lasting attacks, higher attack concurrency could be another factor contributing to the huge average attack time for the group of privileged users.

Victims

For each attack record in the data set, the target is specified as either an IP address or a Web site URL. We identified 689 unique Web sites and 10,485 unique IP addresses in the attack records.

To understand what types of Web sites were victims of DDoS attacks initiated by TwBooter's subscribers, we manually visited the top 100 Web sites in terms of the overall time being under attack. Although the type of targeted Web sites is quite diverse, ranging from other booters to governmental agencies, the overwhelming majority of targeted Web sites were either game servers or game forums. In addition to the attacks on the two journalists, we noticed two users ordering attacks on several different governmental Web sites. The primary focus was on two Indian government Web sites and the Web site of the Los Angeles Police Department. Collectively, the three Web sites were under attack for a total duration of 142 hours by these two users.

Conclusion

Our analysis of TwBooter's attack infrastructure, customers, and victims support the anecdotal evidence that these services are popular and profitable services that are upgrading their attack capabilities as their user bases expand. This enables this service and others to expand from their original purpose as tools used to gain an advantage against gaming opponents, and they are now used to target a diverse set of victims ranging from gamers to small- and medium-sized government Web sites. We have other leaked data sets from larger booter services, such as Asylum, that indicate they had customer bases in the thousands and have been used to launch hundreds of thousands of attacks a year.

The biggest transformation these services create is a business model in which attackers can rent and share DDoS infrastructure that is managed by the booter service instead of building and maintaining their own dedicated infrastructure, thus reducing both the technical and monetary barriers to launching DDoS attacks.

Acknowledgments

We thank Jose Nazario and the other reviewers for their insightful comments of our earlier related LEET paper. This work was supported by the National Science Foundation under grant 1237076 and a gift from Google.

References

[1] http://softwaretopic.informer.com/asylum-booter-source/.

[2] Sean Gallagher, "Details on the Denial of Service Attack That Targeted Ars Technica: http://arstechnica.com/security/2013/03/details-on-the-denial-of-service-attack-that-targeted-ars-technica/, 2013.

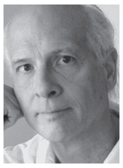
[3] Kelly Jackson Higgins, "DDoS Botnet Now Can Detect Denial-of-Service Defenses": http://www.darkreading.com/attacks-breaches/ddos-botnet-now-can-detect-denial-of-ser/240160466, 2013.

[4] Mohammad Karami and Damon McCoy, "Understanding the Emerging Threat of DDoS-as-a-Service," LEET 2013: https://www.usenix.org/conference/leet13/understanding-emerging-threat-ddos-service.

[5] Brian Krebs, "The Obscurest Epoch Is Today": http://krebsonsecurity.com/2013/03/the-obscurest-epoch-is-today/, 2013.

An Interview with Ben Laurie

RIK FARROW



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

Ben Laurie is a software engineer in Google's security team and a visiting fellow at the Cambridge University Computer Lab. He wrote Apache-SSL, which powers more than half the world's secure Web sites, and he maintains OpenSSL, the world's most widely used open source cryptographic library. Currently he is trying to fix the Internet with Certificate Transparency (http://www.certificate-transparency.org/), among other things. ben@links.org

I had heard of Ben Laurie before, as he is the co-author of several papers I've read, plus open source software that millions of people use: Apache-SSL. And while I was at USENIX Security, Ben's name came up while I was talking about Bitcoin.

I had asked Ben to write in the past, but things never quite worked out, so it seemed natural that I try an interview instead, which did allow me to ask Ben about some things that I was curious about.

Rik: I read your Wikipedia page and learned that you've been working for Google for years. But you are also a Visiting Fellow at Cambridge University's Computer Laboratory, founding director for Apache, a core team member of OpenSSL, security director for The Bunker Secure Hosting, a committer for FreeBSD, trustee and founding member of FreeBMD, and an Advisory Board member of WikiLeaks.org. This seems like an awful lot to be doing while working for Google.

Ben: Indeed I do work for Google, and have done so now for seven years. My current main project is Certificate Transparency [1]. As you say, I am a Visiting Fellow at Cambridge, where I work with Robert Watson's team on a capability CPU, as well as Capsicum [2]. I don't really do much on Apache anymore, but I do work on OpenSSL. And I am a director of The Bunker. Google allows me a portion of my time for some of these things, particularly the work at Cambridge and on OpenSSL. The rest I fit into my copious spare time.

Rik: Reading through your blog, I noticed the post about the CRIME attack on TLS [3]. Was OpenSSL changed in any way to make this attack more difficult? I read in the ARS article that you mention that the popular desktop browsers either were patched or didn't support compression in the first place. Is DEFLATE still supported in OpenSSL? In Apache SSL?

Ben: OpenSSL was not changed nor was Apache-SSL, because Apache-SSL is mostly not used anymore, since mod_ssl (an Apache-SSL derivative) is now included directly in Apache.

Rik: There has been a lot of talk recently (September 2013) about the NSA being capable of "breaking" SSL, and otherwise weakening crypto to make recovering keys easier. With your OpenSSL hat on, does this mean doubling the key length, which will cost big providers a lot of compute time? Or is it really too soon to tell?

Ben: Actually, the CA/Browser Forum has recently doubled key length. From January 2014, SSL keys will have to be 2048 bits.

As for the claim, I don't really believe the NSA relies much on weakening crypto—why would they, when systems are so easy to break into? In my opinion, this whole crypto thing is a complete red herring. We should be focusing on operating system and application security.

Rik: And I have another question. Since I read your blog, I know you're working on Certificate Transparency (CT). Could you explain what the goal of that project is and how it would work?

Ben: As we know since the DigiNotar incident [4], it is possible for a CA to not only fail, but fail and be undetected for a considerable time. The goal of CT is to make it very hard to get a fake certificate and hide that fact. It works by creating a publicly verifiable log of all issued certificates. The idea is that browsers will refuse connections that are not secured with a logged certificate, and domain owners (or their agents) will monitor the log to check that issued certificates [for their domains] are all legitimate.

You can read a lot more here: http://www.certificate-transparency.org/.

Rik: You are also known for the creation of a digital currency. Tell me about what lucre was, and what happened with it.

Ben: Lucre [5] was an implementation of an idea by David Wagner. Essentially, he observed that Chaum's blind signatures, which were patented at the time, were based on RSA, and there was a parallel idea based on DSA. At least, I think it was DSA; perhaps he said Diffie-Hellman.

I implemented this scheme mostly as a learning exercise, but also to explore the costs of issuing digital money. The answer was: very cheap, even back then—much cheaper now, of course. I didn't actually do much with it in practice, though I did implement the whole system as open source. The only serious system I am aware of that was built on it is this: https://github.com/FellowTraveler/Open-Transactions. But I'm not sure how far that got.

Rik: I read "Decentralized Currencies" [6], your article criticizing Bitcoin, and understand your points about Bitcoin's problem with achieving consensus. In your article, you wrote that Bitcoin's proposed consensus group (for deciding where bitcoins reside) is based on "all the computing power in existence." And even though the Bitcoin "proof-of-work" requires more power than the value of the bitcoin produced, the fraction of computing involved in the "consensus group" is actually a tiny fraction of all computing power.

You posit that someone willing to spend a bit more power could produce a longer chain, and thus create a new proof of where bitcoins reside. It seems that you've demolished any logic behind the working of Bitcoin with these two statements, yet people still disagree with you. Could you possibly have this wrong? Is there something I am not understanding about your arguments?

Ben: People disagree with me for two reasons that I can figure out:

a) The amount of work required to make a longer chain is greater than the total work put into the existing chain; that is, to outpace the existing chain you need not just more power, but more power running for as long as the existing power has been. This isn't really an argument against me, so I don't know why people make it, but they do. I guess it's nitpicking at my claim that you need 50% of the total computing power. "Aha!" they say, "you actually need more than that, because we have a head start," which is technically correct, but doesn't invalidate the argument, and the amount you actually need is still far less than half the total, in practice.

b) They sometimes claim that there's no economic incentive to making an alternate chain. There are at least two responses to this: (1) then why are there alternative bitcoin-like currencies? (2) the attacker's incentives may not be economic or the economics may not be the simple economics of bitcoins.

Rik: Sarah Meiklejohn et al. have a paper in IMC 2013 [7] in which they state that 64% of all bitcoins are currently hoarded. In [6], you suggest that Bitcoin is not really a decentralized currency, as the people who created it might be the ones to benefit most from that. We can't tell who is hoarding bitcoins, but is it reasonable to conflate these two ideas?

Ben: Let's not mince words. Bitcoin is a Ponzi scheme. We all know who benefits from Ponzi schemes.

Rik: Since Bitcoin has a fixed cap on the number of bitcoins that can be "minted," doesn't this make Bitcoin a deflationary currency? For me, it recalls the struggle in England between the landed aristocracy, who wanted the currency pegged to land (their land), and those who wanted a currency that could grow as the economy expanded.

Ben: Clearly, the Bitcoin cap favors the Bitcoin aristocracy. As I point out in one of my papers, if you live in the fantasy universe where Bitcoin is decentralized, then there are fairer ways to distribute new, invented wealth (for example, randomly among all the decentralized participants [8]). But in the real world, the wealth of the incumbents rests entirely on the willingness of the next layer of the pyramid to continue to play the game. This is rather different from your historical example, though; there is no land to link the currency to.

Rik: With that out of the way, I have a question about Merkle trees, as that is related to the work you are currently doing. When I read about Merkle trees, the emphasis is on creating a tree of hashes; however, searching for a particular certificate in a tree of hashes boggles my mind, as it implies visiting every node until you find the one you are interested in. Is there a parallel data structure to the Merkle tree for certificates that points to the correct path for the certificate you are interested in? Or, put another way, I am interested in a better understanding of how Merkle trees are used practically, and it seems like you are in a great position to explain this practical use.

Ben: The purpose of the Merkle tree is to allow efficient proofs that

a) the log is append-only,

b) any particular item is in the log (given its location in the log), and

c) everyone sees the same log.

If you want to monitor the log for certificates of interest, you do indeed have to look at the entire log. For certs, that's not such a big deal; we've been up several months now and there's still only 2.5M certs in the log.

Rik: I've read that FreeBSD 9 has shipped with experimental support for Capsicum. I was excited when I first learned about Capsicum in 2010. I recently learned of a new application framework, called Casper, which manages the creation of sandboxes, making it easier for developers to start using Capsicum. What more can you tell us about where this project is going?

Ben: Capsicum is a capabilities system layered on top of POSIX. In short, it adds fine-grained permissions to file descriptors, effectively turning them into capabilities, and a "capability mode." Once a process is in capability mode, which can be done for it by its parent process, it can no longer create file descriptors directly. It can only create copies of existing file descriptors with reduced (or equal) permissions, or acquire new ones from processes it can communicate with; however, it is fully backwards compatible: Capsicum-unaware software can communicate with sandboxed processes, and vice versa (if permitted). It is even possible to "transparently" Capsicumize existing software by replacing libc with a capability-aware version. It is thus possible to gradually migrate from the existing ACL-based world to a capabilities-based one without having to rewrite everything all at once.

FreeBSD 10 will have Capsicum enabled by default, and we're also working on porting Capsicum to Linux [9]. We're gradually Capsicumizing system utilities, but also going after bigger fish, such as SSH, Wireshark, and Chrome.

Casper is a framework and a utility that makes it easier to provide services, such as name resolution, restricted network connections, access to keyrings, and so forth, to sandboxed processes. We're still experimenting with exactly what kind of policies Casper should use, but one of the key aspects of the design is that it is flexible: if you don't like our version, you can replace it with one of your own, either system-wide or for particular purposes. Applications can even provide Casper services directly to each other, with appropriate configuration.

In order to extract maximum value from capabilities, it is necessary to decompose software into separate modules, which, in a POSIX world, necessarily corresponds to processes. We are working on utilities to help with that (http://www.cl.cam.ac.uk/research/security/ctsrd/soaap.html), but we're also looking at a novel CPU architecture to allow fine-grained sandboxing within processes (http://www.cl.cam.ac.uk/research/security/ctsrd/cheri.html). We already have FreeBSD running on this CPU, and a version of Clang that understands capabilities. This allows us to do some pretty cool things. For example, we can modify malloc to return a capability instead of a mere pointer. This completely eliminates heap overflow at a single stroke.

I could write a whole book about the possibilities and advantages, and I invite interested readers to get in touch with us, perhaps on the mailing list (https://lists.cam.ac.uk/mailman/listinfo/cl-capsicum-discuss).

Resources

[1] Certificate Transparency: http://www.certificate-transparency.org/.

[2] Robert N. M. Watson, Jonathan Anderson, Ben Laurie, and Kris Kennaway, "Capsicum: Practical Capabilities for UNIX," 19th USENIX Security Symposium, August 2010: http://www.usenix.org/event/sec10/tech/full_papers/Watson.pdf.

[3] Ben Laurie, "Compression Violates Semantic Security": http://www.links.org/?p=1277.

[4] DigiNotar: https://en.wikipedia.org/wiki/DigiNotar.

[5] Lucre: http://anoncvs.aldigital.co.uk/lucre/; available here: https://github.com/benlaurie/lucre.

[6] Ben Laurie, "Decentralized Currencies Are Probably Impossible: But Let's At Least Make Them Efficient" (critique of Bitcoin): http://www.links.org/files/decentralised-currencies.pdf.

[7] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage, "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names," IMC 2013: http://cseweb.ucsd.edu/~smeiklejohn/files/imc13.pdf.

[8] Ben Laurie, "An Efficient Distributed Currency": http://www.links.org/files/distributed-currency.pdf.

[9] Capsicum code for Linux: https://github.com/google/capsicum-linux.

Mackerel: A Progressive School of Cryptographic Thought

JUSTIN TROUTMAN AND VINCENT RIJMEN

Justin Troutman is a cryptographer with research interests in designing authenticated encryption constructions, optimizing the user experience of cryptographic products, and analyzing methodologies for signals intelligence. He has worked with Microsoft, Google, Duke University, and IEEE. He co-developed the "green cryptography" strategy for minimizing implementation complexity, while maximizing cryptographic security, by recycling components for multiple purposes. He is currently organizing international workshops that explore topics such as optimizing the effectiveness of cryptography for journalists, and the intersection between real-world cryptographic design and experience design.
justin@justintroutman.com

Vincent Rijmen is professor with the Department of Electrical Engineering at the University of Leuven and the security department of the research institute iMinds (Belgium). He is co-designer of the algorithm Rijndael, which has become the Advanced Encryption Standard (AES), standardized by the US National Institute for Standards and Technology (NIST) and used worldwide, e.g., in IPSec, SSL/TLS, and other IT-security standards. Rijmen is also co-designer of the WHIRLPOOL cryptographic hash function, which is standardized in ISO/IEC 10118-3. Recent research interests include security of hash functions, design of methods for secure hardware implementations and novel applications of computer security.
vincent.rijmen@esat.kuleuven.be

Troutman and Rijmen offer a framework for creating and fielding new security systems involving cryptography. Without this or a similar framework to coordinate efforts between different stakeholder communities, we are doomed to continue providing systems that provide less than expected, take too long from conceptual birth to fielding, and still leave us at unacceptable risk.

Their framework is "chunked" to nicely fit the natural flow of effort from cryptographers to developers to users; this allows each profession to maximize its strengths and not be unnecessarily befuddled by work going on in sister professions that are also needed to complete full delivery of effective systems—systems that can be deployed and safely used by customers with minimal training.

Is this work perfect and fully fleshed out? NO. Is it a big step in the right direction? I think YES! Read and enjoy; it is an easy read that should plant interesting concepts in your mind that will take root and grow, leading to further needed developments.

—Brian Snow, former Technical Director of Information Assurance at NSA.

Cryptography is hard, but it's the easy part. It's an entanglement of algorithms and assumptions that only a cryptographer would find poetic, and we're at a point where strong cryptography is arguably the most robust aspect of a system's security and privacy posture. To a consumer, however, cryptography is still an esoteric sort of black magic whose benefits are out of reach. Developers: If you feel we've dropped the ball on safely implementing cryptography—which we have, and horribly so—this doesn't hold a candle to how pitifully we've failed at interfacing the benefits of cryptography to consumers. Our contribution to potentially solving this problem, dubbed Mackerel, is a design and development framework for developers that's based on the premise that *real-world cryptography is not about cryptography; it's about products.*

First, let's look at a process that works, and with which most of us are familiar: buying and driving an automobile. You decide it's time to buy a new vehicle, so you drive to the nearest car lot of your choice. You're greeted by a friendly salesman who wants nothing more than to put you in a new car that day. He needs to sell and you need to buy, so today might be a double-win for both of you. You tell him that with today's gas prices, you need something that gets good mileage, but that you also need something with decent towing capabilities, since you pull a camper to your favorite campground in the mountains. Oh, and with three kids and in-laws, you need a third row of seating. Using his oracle-like knowledge of vehicle statistics, the salesman walks you over to a sporty, yet eco-friendly, SUV that strikes the right balance for all your requirements. You feel the leather seats, admire the hands-free navigation system, and even take it for a test drive. A credit check and some paperwork later, and you're pulling out of the lot in your brand new set of wheels.

When you sit in the car, you shut the door, and (hopefully) buckle up, then proceed to insert the key into the ignition switch, turn it, and the process of internal combustion automagically happens before you. You shift into the appropriate gear, press your foot on the accelerator pedal, and you're off. At no point did you have to understand the mechanics of the vehicle or the process of internal combustion; you simply had to insert a key, turn it, shift a knob, and step on a pedal. In front of you are several indicator lights that give you visual and aural cues that something needs attention. It lets you know if you're about to run out of gas, need an oil change, or if it's something that you should probably have a mechanic check out ("check engine"). You're able to thoroughly enjoy and benefit from the wonder of the automobile, without understanding the physics or mechanics; at the most, your experience as a user involves limited engagement with an intuitive user interface.

So, how can the cryptographic process learn from the consumer automobile experience? Well, we've stated that in order to properly realize the benefits of cryptography as a product, we need to employ the right process—one that respects the roles of people involved. These aforementioned people make up three groups: cryptographers, developers, and consumers. The first mistake, and the cardinal sin, is trying to get everyone on the same page. It's a true exercise in futility because cryptography looks different as it flows from cryptographer to developer to consumer. They each assume distinct roles that require different types of expertise. Ideally, what we want is a process that respects these roles and doesn't ask them to make decisions outside of their realm of expertise. Tragically, it rarely ever happens this way, and we devolve into a modern-day Tower of Babel, trying to collectively build something without having a clue as to what the other is saying. To remedy this, it's paramount that we notice the two relationships that exist here—cryptographer-to-developer and developer-to-consumer—where keeping a tight bond between the former is necessary for *underlying implementation assurance* (think mature and minimalist API), while doing so for the latter is necessary for *user interface accessibility* (think tactile and palatable GUI).

Cryptographer-to-Developer Relationship

Let's start with the cryptographer-to-developer relationship. Cryptographers need to approach developers with a particular golden rule in mind: *cryptographic implementations usually fall apart at the implementation level, not at the cryptographic level.* What this really means is that cryptographers need to create and promote a more benign surface for developers. It's not just about making it easy to get things right; it's even more about making things hard to get wrong. One way to achieve that is through what we call "green cryptography" [1, 2, 3] (extended drafts at justintroutman.com), which calls for the recycling of mature and minimalist components whenever and wherever it makes sense; for example, you can do authenticated encryption (and you should always be doing both authentication and encryption) with a single primitive, like the Advanced Encryption Standard (AES), by using Counter mode (CTR) for encryption and Cipher-based Message Authentication Code (CMAC) for authentication. Or even easier to implement would be an Authenticated Encryption with Associated Data mode (AEAD), which handles both encryption and authentication without the need for two separate modes. EAX (Encryption and Authentication with Associated Data), for example, is essentially a combination of CTR and One-key Cipher Block Chaining Message Authentication Code (OMAC1; equivalent to CMAC), but doesn't require that you manually combine CTR and CMAC; EAX kills two figurative birds with one stone. Not only that, but this particular construction gives you two of the strongest notions of confidentiality and integrity that we have: indistinguishability against Adaptive Chosen-Ciphertext Attacks (IND-CCA2) and Integrity of Ciphertexts (INT-CTXT). Here's a memo you should never say you didn't get: the order of encryption and authentication matters, and it follows that encrypting the plaintext, first, then authenticating the resulting ciphertext, second, is the easiest to get right, hardest to get wrong, and comes with the tightest notions of confidentiality and integrity.

There have also been recent attempts to build cryptographic APIs for developers that make things easier for developers to *safely* implement, such as Keyczar [4] from Google's security team. It achieves this safety by choosing secure default parameters (e.g., block ciphers and key lengths), and automatically taking care of consequential things such as key rotation and IV generation; this is that "benign" surface we mentioned earlier. And speaking of implementation failure as the likely center of catastrophe, there's a class of attacks that preys on the actual software and hardware implementations of cryptography, dubbed "side-channel attacks," in which everything from timing differences to power fluctuations can leak information about plaintext and keys. Fortunately, there's a library with side-channel attack resistance in mind called NaCl (a reference to cryptographic "salt"); with NaCl [5], although you can use standards such as the AES, you have the option of using Daniel J. Bernstein's own cryptographic primitives, such as the fast stream cipher Salsa20 [6], for encryption; there's also the secure message authentication code (MAC), called Poly1305-AES [7], which, although specified for the AES, can be used with other drop-in replacement ciphers. Keyczar and NaCl are important steps toward safer implementations, but they are far from ideal and represent an inch in the miles we need to go. Only by strengthening the relationship between cryptographers and developers can we get there.

Developer-to-Consumer Relationship

Now, let's tackle the developer-to-consumer relationship. Cryptographic software is the quintessential martyr of usability deprivation, a Rube Goldbergian gauntlet of epic distortion. (For our readers from the UK, "Heath Robinsonian." In fact, that's probably the most appropriate name to use, given the cryptographic context.) In [8], they capture most of the reasons why PGP (Pretty Good Privacy) and, by extension, its open source cousin, GPG (GNU Privacy Guard) share the role as poster children for tremendously useful ideas that, although used fervently by some, elude the majority of consumers because of their lack of tactility and palatability and by asking consumers to make configuration decisions far outside their expertise. To be fair, PGP was as novel as it was timely, because at that instant, back when cryptography was a munition, we finally had something that didn't previously exist: a way to keep our email conversations secure and private, with strong cryptography. The point is that it predated the era of usable security and privacy research, and to this day, we still haven't improved much on making it easy to benefit from cryptography. Having said that, we have made strides in recent years when it comes to mediating the marriage of usability with security and privacy tools; in fact, there are academic laboratories focused on it (e.g., Carnegie Mellon's CyLab Usable Privacy and Security Lab, or CUPS) and conferences dedicated to it (e.g., Symposium on Usable Privacy and Security, or SOUPS). These are pioneering efforts that must exist, and we're better for them; on the other hand, cryptography is such a niche subset of security and privacy, and the focus of only a minute portion of this research.

In actuality, to channel [9], "usable cryptography" is as much of an oxymoron as it is manifest destiny; in fact, it's the benefits of cryptography that we should strive for as manifest destiny. Cryptography, itself, as a usable thing, doesn't exist; the utility of cryptography and the usability of a product that implements cryptography exist on entirely different planes. "Usable cryptography" is akin to saying "usable internal combustion." Consumers don't want internal combustion; they want to drive. Just like internal combustion, cryptography is an implementation detail that shouldn't be exposed to the consumer. That's right, consumers rarely, if ever, want cryptography directly; they need what it provides, but that's an entirely different problem. What the consumer actually wants is a useful product, where usefulness ("what am I getting out of this?") is determined by utility ("what does it do?") and usability ("how easily can I do it?").

To exhibit Mackerel as a philosophy for guiding product design, imagine that you're a journalist working under turbulent conditions, in an oppressive environment, and you need to communicate securely and privately with your source; you need an app for your smartphone, and such an app must optimize tactility and palatability, by focusing on: (1) zero learning curve (works with little to no training), (2) rapid-fire accessibility (works intuitively and like the apps you're used to), and (3) minimal code footprints (to simplify, and encourage, third-party auditing). A high-level API could be used to abstract away low-level components, while being conscious of side-channel attacks. Such an API could rest inside of a tactile and palatable GUI that caters to the desires of the user, without exposing you to the complex internals. Ultimately, you need to talk; you need to do it quickly; and, you need to do it easily. It's imperative that the design enables you, not hinders you. If we expose the cryptography to you, we're creating a barrier between the app and what you really want to do. Although you need what cryptography provides, it can't get in the way of you doing your job.

What We Need

We don't need better encryption; we need a better experience. As renowned experience designer Aral Balkan captured in his talk for Thinking Digital 2013, "Superheroes and Villains in Design": as users, we should approach design naively and let it tell us how it wants to be used. When we do this, we recognize the product for what it is, the expert; we should be able to trust it to make the right decisions and give us the affordances we expect. In the case of the journalist above, this implies several things about the user experience. Everything matters. You'll need to consider the right background and foreground colors, and typefaces as well, to prevent eye fatigue from straining to see what's being displayed. Also, you'll have to think about the average size of fingertips so as to prevent misfires; seconds lost to poor interaction can be costly. Oh, and the arrangement of objects on the display is a big deal, too; an object's function should be obvious. And then there's the fact that this journalist is likely to be in vastly different cultures. With that in mind, the symbols and colors you use must make sense within the context of the culture with which the source identifies.

The design should anticipate the needs of its users; the experience should fulfill their wants. The journalist doesn't want to encrypt and authenticate the data channel between himself and the source; the journalist wants to safely talk to his source. He needs the former, but wants the latter. Balkan's forthcoming project, Codename Prometheus, is focused on experience design in the consumer space, with a strong emphasis on protecting security, privacy, and human rights. This is a big step in the right direction of cultivating the experience for the consumer and solving the conceptual problems they care about (e.g., how can I communicate conveniently, but safely?), without burdening them with our own problems regarding the details (e.g., how can I make this app encrypt and authenticate communications?).

What all of this is trying to tell us is that we've been taking a monolithic approach to development for far too long. It's simply not enough for cryptographers to help developers properly implement; that's only one-half of real-world cryptographic design. What we absolutely must have are experience designers helping developers properly interface. Ignoring this carries on the tired, hapless campaign of "cryptography for the masses," which didn't materialize into the cypherpunk dream; by inviting it, however, we have a fair shot at helping those masses benefit from cryptography. In the cryptographer-to-developer relationship, cryptographers have the ability to work with developers on this problem; in the developer-to-consumer relationship, the consumer hasn't the expertise to work with the developer. Experience designers do, however; they speak to the needs and wants of the consumer on their behalf. In other words, developers have a chance at getting the implementation right with cryptographers around; without experience designers around, however, there's little hope of them getting the interface right.

A Fish Called Mackerel

Mackerel is a cryptographic design paradigm that posits that practical cryptography is essentially a subset of product design. And because it's about products, it's about people, and the need for a holistic product design process that respects the roles of the people involved—cryptographers, developers, and consumers—by only asking them to make decisions that lie within their respective areas of understanding, and of which they understand the consequences. Ultimately, by focusing on the cryptographer-to-developer and developer-to-consumer relationships, the outcome will render the assurance of the underlying implementation, as well as the accessibility of the user interface, resulting in a product that's useful, by offering both utility and usability to the consumer, and that behaves securely and privately. In short, Mackerel is a developer-centric, consumer-targeted "conception-to-cellophane" approach to building a cryptographically enhanced product from the ground up; the goal is to optimize the GUI (interface accessibility) and API (implementation assurance), by looking at tried-and-true elements from both product design and security engineering.

The Mackerel framework is intended to operate similarly to a software development framework, where the design and development of a cryptographic product is modeled as a dissection of individual components that, although they all affect the overall goal of security and privacy, often require distinct approaches. For example, within this framework would be cryptographic threat modeling, where the intended application of a product and its operating environment are considered in order to determine applicable attacks and the appropriate cryptographic measures for mitigating them. This is clearly a security and privacy problem with a security and privacy answer; however, as the framework shifts from low-level to high-level, where you're dealing with usability factors and the overall experience of the product, you're dealing with a problem that can't be answered by security and privacy experts. (If we try to do so, we risk PGP 2.0: hard to break, but hard to use.) It can be answered by usability experts and those who design experiences for a living, which is what has been missing in the modern day process. Although a bad interface and experience can lead to a poor security and privacy decision, this doesn't mean the interface and experience are security and privacy problems or can be solved as such; it means that we can't solve the interface and experience problems without experts in those areas working alongside security and privacy experts. We currently having nothing of the sort, let alone a framework that involves both.

Once you birth cryptography into the real world, it becomes a small component in a large composite that has more non-cryptographic parts than cryptographic ones; having said that, you can't build a good cryptographic product if you involve cryptographers but not product designers. You certainly can't build a good cryptographic product if you think it's entirely a cryptographic problem, or even entirely a security and privacy problem. Mackerel models every core aspect of cryptography's evolution as a product, such that optimal decisions can be made, given the state-of-the-art know-how in cryptographic design, software development, and user experience design.

Lastly, let's tell you why Mackerel is called "Mackerel." At first glance, it might seem like just another entry into cryptography's long list of systems named after fish. Well, that's partially true, but there's a bit more. Integrity is as important a goal as confidentiality, if not sometimes more. After all, breaking confidentiality is the ability to passively eavesdrop, whereas breaking integrity is the ability to actively manipulate. You can imagine how the latter can render far worse results than the former, and even result in the loss of both. So, although encryption is supposed to handle confidentiality, it often can't, without authentication, and the standard way to go about that is through the use of a MAC, or message authentication code. If there's anything out of all of this research that we hope you learn, from a cryptographic point of view, it's that you should always use a MAC, or an AEAD mode that does both encryption and authentication, or die trying.

In order to pay homage to the glorious yet underappreciated MAC, it was befitting to choose as a moniker the fish whose name begins with "mac": the mackerel.

Acknowledgments

We immensely thank the vast number of people whose eyes and ears have been so graciously loaned over the past five years since our work on "green cryptography" first emerged. A distinguished thanks to Brian Snow for supporting our vision and for helping to shape it with his uncanny know-how and grasp of the level of assurance and accessibility we should expect from a cryptographic product.

References

[1] J. Troutman and V. Rijmen, "Green Cryptography: Cleaner Engineering through Recycling," *IEEE Security and Privacy*, vol. 7 (2009), pp. 71-73.

[2] J. Troutman and V. Rijmen, "Green Cryptography: Cleaner Engineering through Recycling, Part 2," *IEEE Security and Privacy*, vol. 7 (2009), pp. 64-65.

[3] J. Troutman, "Green Cryptography": extended drafts can be found at http://justintroutman.com, 2013.

[4] A. Dey and S. Weis, "Keyczar: A Cryptographic Toolkit," 2008.

[5] D. J. Bernstein, T. Lange, and P. Schwabe, "The Security Impact of a New Cryptographic Library," Cryptology ePrint Archive, Report 2011/646, 2011: http://eprint.iacr.org.

[6] D. J. Bernstein, "The Salsa20 Family of Stream Ciphers," in *New Stream Cipher Designs* (Springer-Verlag Berlin, 2008), pp. 84-97.

[7] D. J. Bernstein, "The Poly1305-AES Message-Authentication Code," in *Fast Software Encryption* (2005), pp. 32-49.

[8] A. Whitten and J. D. Tygar, "Why Johnny Can't Encrypt," Proceedings of the 8th USENIX Security Symposium, 1999.

[9] M. E. Zurko and A. S. Patrick, "Panel: Usable Cryptography: Manifest Destiny or Oxymoron?," in *Financial Cryptography* (2008), pp. 302-306.

SYSADMIN

Trusting PGP

PHIL PENNOCK

Phil is a software engineer at Apcera, provider of the modern enterprise IT platform. phil.pennock@spodhuis.org

PGP is a great tool, but if you're coming to it now, after this year's NSA revelations, then it's probably not the service you want. In fact, I'll go further: if PGP is being peddled to you as the panacea to the NSA issues, the peddler probably doesn't understand what they're talking about.

In all security decisions, you should decide what you're trying to protect and from whom. Additionally, you should decide how much the protection is worth to you. Only once you've done this, can you decide which attributes (confidentiality, authenticity, etc.) you need and what tradeoffs are worth it.

For various good reasons, I run my own mail service that serves only two people; for various other reasons, I stand out like a sore thumb. Frankly, the NSA is not in my threat model. If it were, I wouldn't run servers with network services provided by programs written in C. In this article, I assume that the reader is dealing with people who have suddenly decided that the NSA is part of the threat model and that the reader needs data points to apply in a re-education process.

Traffic Analysis

A number of actions have driven folks to look for more privacy, but the core of the movement lies in that word, "privacy," and the NSA's wholesale gathering of traffic analysis data, of everyone everywhere always. PGP can help you with everything after that initial traffic analysis gathering. Traffic analysis is all about knowing who is talking to whom, and when. PGP, as an object-level privacy wrapper, not only does not hide that, it actually embeds the keys of the recipients into the message. This is optional, but almost always done, because it makes life easier for the recipients when there is information in the wrapper about which keys were used as part of encryption. These are the recipient keys that are provably tied to a given identity, if you've gone so far as to arrange for trust verification.

Unfortunately, most mail clients with PGP integration do a rather poor job of managing Bcc recipients; too often, the keyIDs of all the recipients are listed in the wrapper. If you want to send encrypted email and use Bcc, do some testing first before trusting the integration. Ideally, the Bcc'd copies will be sent as independent SMTP transactions.

If you're trying to avoid traffic analysis while still using email, then I suggest that you hide in the anonymity of crowds. That means using a mail service that provides mail for many people, not just you. If the mail service you choose uses SMTP/TLS for MX delivery, then all that an eavesdropper knows is that someone in domain A sent a message to someone in domain B. You can get a lot of protection, if you trust the mail service provider to provide privacy up until an individual legal warrant is served if both users are in the same domain, and there are enough users that the timing of the connections won't reveal anything, and neither will the sizes of data transferred. If domains A and B are large and use TLS between each other, you've doubled the number of service operators to be trusted but are still protected from traffic analysis because of the sheer volume of data continuously being exchanged between the two.

Naming
OpenPGP is the technical name for the standards, GnuPG is a common implementation of that, as is pgp(1) from the company PGP, but most commonly the systems are simply called PGP, the name of Phil Zimmermann's original implementation.

For mail service protection against traffic analysis, the bigger the provider, the better. If your provider offers SMTP and IMAP access, you can still use PGP to protect the content instead of having to trust the provider. Of course, if most people are using Webmail interfaces, then you risk standing out.

Using PGP Safely

With that out of the way, there are issues to understand around using PGP safely. After key selection and key sizes (just use RSA with 3072 or more bits; 4096 is the practical upper bound [1]), it all comes down to key identity, knowing which keys are correct and how you retrieve the keys to use.

PGP uses the Web of Trust model, with each client making its own trust decisions. Each PGP key is a self-contained certificate, and implicitly an always-open Certificate Signing Request; the "key" passed around is a bundle of collected signatures each made by various other people's keys. Anyone can sign anyone else's keys. Be aware that here is some confusing terminology: a PGP "key" contains a cryptographic public "key" and attestations of identity, which various people will certify.

As an example, let's work with Alice, Bob, Charlotte, and Derek. Alice meets Bob, exchanges enough information with him about his key to decide that the key she has for him really is his public key. Perhaps Bob has his key fingerprint on his business card. Bob similarly verifies Charlotte's key, and Charlotte does the same with Derek. Now, if Alice can trust Bob to do a good job, and Charlotte to do a good job, despite never having met Charlotte, then she might be able to trust that she has the "right" self-contained certificate matching an identity for Derek to a public key. She might have trust in the chain.

Fundamentally, if Bob gives me his complete PGP key, I trust that the cryptographic public key contained therein is his. I might not yet trust the email address claimed in the key, or that Bob's name really is Bob, but I'll trust that the public key material presented is Bob's. Issuing a public signature is about verifying who Bob is and whether he really does own the email addresses that he claims to own.

This attestation of identity doesn't scale well, because People Are Lazy. We all know people are lazy. Anyone who has tried to get PGP adopted more widely will have dealt with folks who will sign any key, and publish that signature, not caring that they've made a public attestation of identity, saying "trust me on this," without bothering to do any checking to back that attestation. There's a reason that our society has the concept of public notaries: a possibly unfair subset of the population whom we might choose to trust to bother doing checking. And heck, we might even trust the people choosing those notaries to do an honest job.

Sure, there are some people you might trust. Sometimes just seeing the email domain is sufficient to infer something meaningful. For instance, even though I don't use Debian, I trust their training and indoctrination enough that when, during a trust database update, I'm presented with an @debian.org UID, I'll usually choose to score the key as having "marginal" trust instead of "don't know," even if I don't know who the person is.

Given that People Are Lazy, the first and biggest problem here is that the default mode for PGP clients always seems to be to create public signatures when signing someone else's key. Each key signature you make can either be "exportable," that is, public, or "local," used purely for personal convenience. The distinction is purely a Boolean in the PGP data structure of the signature, used as a hint that the signature should not normally be exported for use by others. I'll posit that most users never think through the issues enough to develop a viable mental model of the tools and concepts they're dealing with, so the default should probably be to make local signatures, with a --tell-others-to-trust-me-on-this flag to create an "exportable" signature. That simple act might encourage others to gain enough understanding to make the Web of Trust look less like a web woven by a spider on meth. Fewer signatures might appear to hurt scaling, but they'd be higher quality signatures by default.

If you want to convey more information to others about the verification you have done, PGP lets a key signature include a policy URL to provide a pointer to a description of the sorts of verification done. With GnuPG, you can use the --cert-policy-url option to set this.

Key Distribution

Who are you trusting, with what information, when you fetch a PGP public key? Are you using the public keyservers? I run one of those—I have patches in the codebase—and I think they're useful enough that I'll continue to do this as a public service for as long as I think it tenable. But public keyservers are also filled with junk. Even spam. The public keyservers do no trust-path verification. They barely manage to check that a key is valid or well-formed. They should be rejecting non-exportable key signatures, but the main peering mesh of keyservers doesn't do that and keyserver developers are trying to figure out what to do about that without breaking the key set reconciliation algorithm. Presence of a key in the public keyservers means nothing.

Furthermore, when you ask a keyserver for a key, you're communicating to that keyserver who or what it is you want to communicate with; whether a human for email, or a Web server participating in Monkeysphere. This provides information for the same traffic analysis discussed earlier. The people maintaining pool definitions of public keyservers don't validate the people running the keyservers, they just validate that there's a functional keyserver that is staying up-to-date with "enough keys that it's probably current." A spider does this validation by walking the stats pages of the keyservers, figuring out what the

Keysigning Parties

The security of PGP is largely built around being able to validate identity assertions via the Web of Trust. Although anyone can validate another's key at any time, there are scaling efficiencies to organized events. You will often find in the schedule for technical conferences a BoF type session that facilitates mass cross-signings. It's called a "party," but that is surely someone's idea of a joke. You only need to attend one or perhaps two of these events to have your key end up in the "Strong Set," about which you can find more details online.

In short, all folks planning to attend let the organizer know ahead of time, with their keyIDs. The organizer prepares a keyring with those keyIDs and prints out sheets of paper with each key and fingerprint on it. Each attendee receives a copy of that at the event. During the event, some mechanism will be used to let each person see every other person's photo ID; depending upon attendee count, that might be passing cards around, or a somewhat sophisticated conga line that will eventually dissolve into chaos. But it's still not a party. Then each attendee in turn will stand up and read out their key fingerprint, from their own trusted source (not the paper copy prepared by the organizer), letting each other attendee check off the details. By the end of this, each attendee should have some confidence in a legally accepted human name attached to each keyID. Verifying the email addresses is then a matter of sending the signature of each PGP UID to the address in that UID, encrypted to the key, shifting the responsibility to the key-owner to upload the key to public keyservers. The two main tools to automate this are "caff" and "pius" [2]. This means that the extent of the email verification is "someone with access to the mailbox also had access to the private key and was happy to affirm the association."

peering mesh is, and determining which keyservers can be used to get current keys. The spider then updates DNS to update the records returned for various pools, including geographic pools.

If intelligence agencies aren't running public keyservers under hostnames designed to sound cypherpunkish, to get a percentage of traffic analysis about who wants to talk securely with whom, then they're slipping. They could skip this step, as the communication between keyservers is an old protocol using HTTP with a fixed pattern of URL construction, no matter which host is chosen. This protocol is called HKP, which stands for Horowitz or HTTP Keyserver Protocol, depending upon whom you ask. The traffic is almost always unencrypted. A well-placed traffic tap for data flowing to and from the default HKP TCP port, 11371, is probably very informative.

The non-keyserver approaches usually involve tools such as finger, also unencrypted.

If you want people in your organization to have some privacy in whom they're communicating securely with (end-to-end), consider running a local "SKS" keyserver: you'd currently need to also provide this as a public service as an inherent part of how you exchange traffic with peers, but the front-end HTTP proxy you put up can also offer HTTPS (HKPS) communication on a known hostname, so that there's an identity your software clients can validate, using the PKIX, which is an entirely different can of worms. If you have an internal certificate authority, and manage internal software deployments enough to control default GnuPG configurations, you can at least ensure that only your CA is trusted for keyserver host identities.

There is little HKPS in the public PGP keyserver web because most client communication is via pool hostnames, and getting PKIX signatures for pool services run by unaffiliated independent groups certainly should be impossible. And even if you had that, it would be incentive for some well-funded groups to offer keyservers that happen to forward logs of retrievals to some (perhaps local) acronym agency.

There are no better solutions for OpenPGP on the horizon if you're concerned about traffic analysis. Various systems that put keys into DNSSEC-secured DNS can provide for confidence that you have the right key, but certainly don't protect against a network traffic filter for DNS traffic concerning the CERT or OPENPGPKEY RR-types.

Anything using well-known URLs or services in the recipient's domain will leak some traffic data when you retrieve the key; in the best-case scenario, where the link is encrypted, it's obscured to the domain level. In the worst-case, you've just signaled to the world that now would be a good time to compromise your client machine.

Good luck helping your paranoid users thump hard back into reality.

References

[1] For more guidance on choosing key lengths, see http://www.keylength.com/en/3/, the "Asymmetric" table column. This site provides guidance from several sources, so you can pick and choose depending upon whom you trust most for advice.

[2] Tools to help with keysigning parties: pius, http://phildev.net/pius/; caff, http://pgp-tools.alioth.debian.org/.

11th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’14)

Sponsored by USENIX, the Advanced Computing Systems Association

October 6–8, 2014, Broomfield, CO

Important Dates

Abstract registration due: *Thursday, April 24, 2014, 9:00 p.m. PDT*

Complete paper submissions due: *Thursday, May 1, 2014, 9:00 p.m. PDT*

Notification to authors: *Thursday, July 17, 2014*

Final papers due: *Wednesday, September 10, 2014*

Symposium Organizers

Overview

The 11th USENIX Symposium on Operating Systems Design and Implementation seeks to present innovative, exciting research in computer systems. OSDI brings together professionals from academic and industrial backgrounds in what has become a premier forum for discussing the design, implementation, and implications of systems software. The OSDI Symposium emphasizes innovative research as well as quantified or insightful experiences in systems design and implementation. OSDI takes a broad view of the systems area and solicits contributions from many fields of systems practice, including, but not limited to, operating systems, file and storage systems, distributed systems, cloud computing, mobile systems, secure and reliable systems, embedded systems, virtualization, networking as it relates to operating systems, management and troubleshooting of complex systems. We also welcome work that explores the interface to related areas such as computer architecture, networking, programming languages, and databases. We particularly encourage contributions containing highly original ideas, new approaches, and/or groundbreaking results.

Submitting a Paper

Submissions will be judged on originality, significance, interest, clarity, relevance, and correctness. All accepted papers will be shepherded through an editorial review process by a member of the program committee.

A good paper will:

- Motivate a significant problem
- Propose an interesting, compelling solution
- Demonstrate the practicality and benefits of the solution
- Draw appropriate conclusions
- Clearly describe the paper's contributions
- Clearly articulate the advances beyond previous work

All papers will be available online to registered attendees before the conference. If your accepted paper should not be published prior to the event, please notify production@usenix.org. The papers will be available online to everyone beginning on the first day of the conference, October 6, 2014.

Papers accompanied by nondisclosure agreement forms will not be considered. All submissions will be treated as confidential prior to publication on the USENIX OSDI '14 Web site; rejected submissions will be permanently treated as confidential.

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

Prior workshop publication does not preclude publishing a related paper in OSDI. Authors should email the program co-chairs, osdi14chairs@usenix.org, a copy of the related workshop paper and a short explanation of the new material in the conference paper beyond that published in the workshop version.

Questions? Contact your program co-chairs, osdi14chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

By submitting a paper, you agree that at least one of the authors will attend the conference to present it. If the conference registration fee will pose a hardship for the presenter of the accepted paper, please contact conference@usenix.org.

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

Deadline and Submission Instructions

Authors are required to register abstracts by 9:00 p.m. PDT on April 24, 2014, and to submit full papers by 9:00 p.m. PDT on May 1, 2014. These are hard deadlines. No extensions will be given. Submitted papers must be no longer than 12 single-spaced 8.5" x 11" pages, including figures and tables, plus as many pages as needed for references, using 10-point type on 12-point (single-spaced) leading, two-column format, Times Roman or a similar font, within a text block 6.5" wide x 9" deep. Final papers may gain two pages, for a total of 14 pages. Papers not meeting these criteria will be rejected without review, and no deadline extensions will be granted for reformatting. Pages should be numbered, and figures and tables should be legible in black and white, without requiring magnification. Papers so short as to be considered "extended abstracts" will not receive full consideration.

Papers must be in PDF format and must be submitted via the Web submission form on the Call for Papers Web site, www.usenix.org/osdi14/cfp.

Blind reviewing of full papers will be done by the program committee and external review committee, with limited use of outside referees. Authors must make a good faith effort to anonymize their submissions, and they should not identify themselves either explicitly or by implication (e.g., through the references or acknowledgments). Submissions violating the detailed formatting and anonymization rules will not be considered for publication.

Authors are encouraged to contact the program co-chairs, osdi14chairs@usenix.org, if needed to relate their OSDI submissions to relevant submissions of their own that are simultaneously under review or awaiting publication at other venues. The program co-chairs will use this information at their discretion to preserve the anonymity of the review process without jeopardizing the outcome of the current OSDI submission.

For more details on the submission process, and for templates to use with LaTeX, Word, etc., authors should consult the detailed submission requirements linked from the Call for Papers Web site.

Jay Lepreau Award for the Best Paper

The program committee will, at its discretion, determine which paper(s) should receive the Jay Lepreau Award for the Best Paper.

Poster Session

We plan to hold a poster session in conjunction with a social event at the Symposium. Details on submitting posters for review will be available on the Web site by August 2014.

Registration Materials

Complete program and registration information will be available in August 2014 on the conference Web site.

Rev. 10/25/13

Using SEC

DAVID LANG

David Lang is a Staff IT Engineer at Intuit, where he has spent more than a decade working in the Security Department for the Banking Division. He was introduced to Linux in 1993 and has been making his living with Linux since 1996. He is an Amateur Extra Class Radio Operator and served on the communications staff of the Civil Air Patrol California Wing, where his duties included managing the statewide digital wireless network. He was awarded the 2012 Chuck Yerkes award for his participation on various open source mailing lists.
david@lang.hm

As you build your enterprise logging infrastructure (as discussed in the prior articles in this series [1]), one of the most valuable things that you can do is to have something watch them and generate alerts when things go wrong. There are a lot of tools out there that can be used for this. One good, free tool is Simple Event Correlator (SEC) [2]. In this article, I will provide an introduction to SEC, how to use it, and the capabilities that it provides. In a future article, I will go into detail about how to tune your SEC installation to be able to handle high volumes of logs.

SEC can read log files directly on the local system, but in the context of an enterprise logging infrastructure, this is seldom the right thing to do.

Instead, because you have a consolidated feed of all your logs, you should run one or more instances of SEC on a central analysis farm server where it can see the logs from all your different systems. This allows you to create alerts for things that happen across multiple servers. For example, you don't want to alert on one failed login, but one failed login to each of 400 servers is something that you do want to be alerted about. Additionally, there is a lot of value in keeping your configuration all in one place so that the same rules will be applied across all systems.

Of course, with advantages come disadvantages. The fact that you see the logs from all your systems means that your alerts will fire no matter what environment your systems are in. You probably don't really want a wake-up call at 3 a.m. because a Dev or QA system had a problem, whereas you would want such a call if it was a production system. This is why the enterprise logging architecture defined a way to add metadata to the log messages. Among other benefits, this metadata provides your alerting farm more information than just what is in the log messages when deciding whether it should generate an alert.

The best way to feed log events into SEC is to make sure you are using SEC 2.7.4 or newer and then use the omprog output of rsyslog to have rsyslog start SEC (restarting it, if needed).

With rsyslog 7, this would be done with configuration lines like:

```
Module (load="omprog")
action(type="omprog" binary="/usr/sbin/sec --input=- --initevents
   --notail –conf=/path/to/conf" template="RSYSLOG_TraditionalFileFormat")
```

With older versions, the configuration would be something like:

```
$ModLoad omprog
$ActionOMProgBinary /usr/local/bin/sec.sh
*.* :omprog:
```

and you would have to define the script /usr/local/bin/sec.sh to be something like:

```
#!/bin/sh
/usr/sbin/sec --input=- --initevents --notail –conf=/path/to/conf
```

Understanding the SEC Config File

Sample Rule

SEC configuration consists of multiple rule definitions, along the lines of:

```
type=single
ptype=regexp
pattern= (\S+) sshd\[\d+\]: Accepted.*for (\S+) from (\S+) \
port  (\d+)\s
desc=ssh login to $1 from $3 for user $2
action=write - $2 logged in to $1 from $3 port $4
```

This rule would look for a line like this:

```
Sep 16 17:46:47 spirit sshd[12307]: Accepted password for rik from
                    204.176.22.9 port 59926 ssh2
```

And when it is found, would generate an alert to stdout that said:

```
rik logged in from spirit IP 204.176.22.9 port 59926
```

Notes on syntax:

- The order of the keyword=value clauses within a rule does not matter.
- Keywords are case sensitive (unless otherwise specified in the man pages).
- Lines can be continued by ending them with a \.
- Rules are separated by blank lines.
- Comment lines start with #, because comment lines are treated by SEC as if they were blank; comments cannot appear in the middle of a rule but must be between rules.

Many of the values that you are providing are sensitive to case and whitespace. For example, the pattern provided in the code above is looking for a space ahead of the hostname.

Type

There are many different types of matches that SEC has built-in:

SINGLE

If a match is found, take action immediately.

SUPPRESS

Ignore anything that matches.

CALENDAR

Cron type rule to take action at specific times.

SINGLEWITHSUPPRESS

If a match is found, take action immediately and suppress additional alerts for a time.

PAIR

Watch for pairs of log entries and take one action when the first entry arrives, and a second if the second entry arrives in time.

PAIRWITHWINDOW

Watch for pairs of log entries, take one action if the second event arrives in time, and take a different action if it does not. Unlike Pair, no action is taken when the first entry arrives; an action is only taken when the second entry arrives or the timeout hits.

SINGLEWITHTHRESHOLD

Take action if there are X matches in Y time.

SINGLEWITH2THRESHOLDS

If there are more than X matches in Y time, take one set of actions, and then wait until there are fewer than X2 actions in Y2 time and take another set of actions—i.e., send a notification when a problem happens (too many messages) and a second notification when it clears up (the problem messages disappear).

EVENTGROUP

This rule is a generalization of the SingleWithThresh old rule; instead of counting and thresholding one event type, this rule is able to track unlimited number of different events types in a common window (e.g., generate an alarm if ten firewall events and five IDS events have been seen for the same IP address during one minute).

SINGLEWITHSCRIPT

If a match is found, run a script and take one of two actions depending on whether the script returns success or not.

JUMP

If a match is found, process one or more other config files against this event.

Ptype

For each rule, you must tell SEC which of the many possible pattern types this rule is using. The available pattern types are:

1. RegExp: Perl regular expression

This pattern type can set variables based on match terms; items enclosed with () in the regexp become $# variables for the rest of the rule. So the sample rule example sets four variables:

```
pattern= (\S+) sshd\[\d+\]: Accepted.*for (\S+) from (\S+) port
(\d+)\s
Sep 16 17:46:47 spirit sshd[12307]: Accepted password for rik from
        204.176.22.9 port 59926 ssh2
```

$1 hostname (spirit),

$2 username (rik),

$3 source IP (204.176.22.9),

$4 port number (59962).

Plus the default $0, which refers to the entire line.

2. SubStr: Substring

Substrings are simple text matches, have no special characters like a regular expression, and don't return any values ($1, $2…). They are much faster to process than a regexp.

3. PerlFunc: Perl function

This executes a Perl function and match if the function returns true and is an extremely powerful capability that I will talk about more in a later article.

This pattern type can also set variables. The Perl snippet can return a list, and the elements of that list become the $# variables.

4. Cached: uses the results of a prior rule match.
5. Tvalue: either matches everything (TRUE) or matches nothing (FALSE).

Cached and Tvalue are normally combined with context conditions, which are described below.

Each of these pattern types will have a negated version (e.g., NregExp, NsubStr, etc.).

Pattern
Most rules require one (or more) pattern lines, and the syntax of the pattern is defined by the ptype defined for the rules.

Desc
Desc fields are critical to understand when configuring SEC. They seem simple (a description of the match), but they play a critical role when doing anything more than a single match.

Proper use of the desc field allows one rule to run many event correlations in parallel and track the state of the correlations independently. Desc defines a "scope" for the correlation state.

When SEC is evaluating any type that has to look at more than one log entry, SEC considers the desc field to be part of the rule. This means that if the desc field evaluates to a different value for the log event, the scope is different and progress towards generating an alert (or suppressing events after an alert has been generated) will be tracked independently of log events that result in the desc field evaluating to a different value.

So if we were to take the sample rule from above and change it to a SingleWithSuppress rule (we don't want alerts every time someone logs in), the rule would become:

```
type=singlewithsuppress
ptype=regexp
pattern= (\S+) sshd\[\d+\]: Accepted.*for (\S+) from (\S+) \
port (\d+)\s
desc=ssh login to $1 from $3 for user $2
action=write - $2 logged in to $1 from $3 port $4
window=60
```

With this rule, we would only get one alert per minute for the same user logging in to one server from another server.

But if we wanted to change this alert so we only got one alert per minute about the user logging in, no matter what server the user logged in to or where the user came from, we could change the desc field to:

```
desc=ssh login for user $2
```

If we wanted to suppress messages only if the user is logging in from the same source, we could change it to:

```
desc=ssh login from $3 for user $2
```

Note that SEC doesn't actually care what this text is, so it would be just as valid as far as SEC is concerned to have the desc field be:

```
desc=$3 $2
```

But it is much nicer to the humans who have to read the file if you make the field more descriptive. SEC combines the desc field with the rule number, so if you have multiple rules that produce the same desc string, SEC will still keep them straight.

Action
An action is what SEC should do when it finds some condition. A single rule can invoke many different actions, semicolon separated. SEC supports many different actions in several different categories. The more important ones to understand include output actions to let you write to a file, a TCP, UDP, or UNIX socket, or execute a script and pass data to stdin on that script.

These commands all have the form:

```
action=<action> <destination> <string>
```

Especially notable are the udgram and spawn actions.

The udgram action lets you send a message to a UNIX socket like /dev/log, which is a great way to have SEC generate feedback into the logging system that can be acted on by other analysis engines. In an enterprise environment, this is also the best way to generate new events for SEC to process because it will work across multiple instances of SEC, and the event will be visible to all your different analysis farms:

```
action=udgram /dev/log <30>sec-alert: alert text
```

Note that "<30>" is the over-the-wire representation of priority: facility (3 daemon) <<3 + severity (6 info) [3].

The spawn action runs an external program and reads any output from that program as additional log events to analyze.

Context actions let you `create`, `delete`, or redefine (`set`) a context. There are also actions to manipulate and output the list of strings associated with a context (including `add`, `prepend`, `report`, `pop`, `shift`, `copy`).

And, finally, there are actions to set variables. Because it is possible to set a variable to be the output of Perl code, and that Perl code is allowed to have side effects, these actions turn out to be the most powerful.

Additional Important Rule Options

Continue

By default, SEC stops processing a log entry the first time a rule matches that entry. Continue tells SEC whether it should continue processing rules if this rule matches. The default is DontCont, which stops processing rules as soon as one matches the event being processed. By adding a line to the rule that says:

```
continue=takenext
```

SEC will continue processing the rules for the current log entry.

If you wanted to use the different desc examples together—for example, alerting if one user is logging in too many times, or if one machine has too many logins to it—you would need to make sure that the earlier rules all include `continue=takenext` or SEC will never get to the later rules.

Contexts

Contexts (and the desc field described earlier) are the heart of SEC and are what makes it more than simply a fancy regexp engine. Whereas the desc field lets one rule run many event correlation operations simultaneously, and thus act as if it uses many rules, contexts allow you to stitch multiple rules together.

Contexts have four properties:

- Existence—manipulated by the create, delete, obsolete actions
- Defined lifetime—defined at creation or reset by the set action
- Storage—manipulated by the add, . . . actions
- Expiration action—again set during creation or by using the set action and can be used for a number of different things:
- Controlling the actions of other rules by testing to see if a context exists. This allows you to dynamically switch rules on and off by checking for combinations of one or more contexts.
- Storing events and other strings via the add, . . . actions. The stored information can then be reported using the report action.
- Scheduling actions to occur in the future by setting an expire action and a lifetime in seconds.

Contexts are created and manipulated by the action section, but are tested by adding a context= clause to your rule

For example, if you want to alert if you see logs foo, bar, and baz all happen within one minute from the same machine, you could create the rule file:

```
type=single
ptype=regexp
pattern=^.{16}(\S+) .*foo
continue=takenext
action=create foo_$1 60

type=single
ptype=regexp
pattern=^.{16}(\S+) .*bar
continue=takenext
action=create bar_$1 60

type=single
ptype=regexp
pattern=^.{16}(\S+) .*baz
continue=takenext
action=create baz_$1 60

type=single
ptype=regexp
pattern=^.{16}(\S+)
context=foo_$1 && bar_$1 && baz_$1
continue=takenext
action=write - warning foo bar baz on $1; \
  delete foo_$1; delete bar_$1; delete baz_$1
```

Note that this set of tests works even if the logs arrive in a different order than you expected.

Executing Perl code as part of the context test is also possible. When combined with cached pattern types, this allows for specific and fast rule evaluation.

Contexts allow you to combine multiple rules in one alerting decision. You can alert only if several different conditions are true by having one rule for each condition you are interested in (each one setting a context), and then another rule to detect that all of the other criteria have been met.

The most common use of Contexts is to set a flag (with a timeout) so that other rules can know that a particular condition has taken place.

Another use for Contexts is to alert when something stops happening. For example, if you have your systems running “vmstat 60 |logger -t vmstat”, they will log a vmstat output line every minute. You can then use a rule similar to:

```
type=single
ptype=regexp
pattern= (\S+) vmstat:
desc=vmstat_$1
action=create vmstat_heartbeat_$1 180 ( shellcmd notify.sh $1 )
```

to generate a notification whenever a system ($1) stops generating a log message. It does this by creating a context that will expire in three minutes, and if the context expires, it sends a notification. If another vmstat message arrives from that system, SEC resets the context to expire three minutes from when that message arrived.

The ability to associate a list of strings with a context allows you to create a context when you see the first event that makes you suspicious, add all log events as strings to the context, so that when the context expires (or some other condition happens), you can make all of the logs that occurred during this period be part of the alert that you send out.

Internal Events

When started with –initevents (as in the example of how to start SEC from rsyslog), SEC generates internal events as it is running; this allows you to create actions that only take place once when SEC is started, restarted, shutdown, etc. For example, if you want a log entry every time that SEC is started or restarted, you could use a rule like:

```
type=single
ptype=regexp
pattern=(SEC_STARTUP|SEC_RESTART)
context=SEC_INTERNAL_EVENT
desc=Init counters with 0
action=udgram /dev/log <30> sec-status: SEC initialized
```

Using Perl in SEC

The ability to use snippets of Perl in your SEC rules is one of the things that makes SEC so incredibly powerful. SEC runs your Perl snippets in a different namespace than SEC itself, so your Perl snippets are not going to conflict or interfere with the SEC internals, although it is possible to get access to the SEC internal variables if you really need to.

As an example of the capabilities that this provides, you could extend the sample rule above to produce daily ssh login reports by changing the action in the sample rule above to:

```
action=write - $2 logged in to $1 from $3 port $4; \
       eval %o ($ssh_summary{user}{$2}{count}++; \
       $ssh_summary{total_sessions}++; )
```

SEC doesn’t actually have a command only to execute Perl code, but it has actions that allow you to run any Perl code and put the output of that code in a variable. In this case we put the output of the Perl code into the variable %o, but we never use it. The exec action compiles the code each time; there is a similar action lcall that compiles the code once at startup. This is faster, and it can avoid the need to escape Perl variables.

Add a rule to initialize the variables at startup (and restart).

```
type=single
ptype=regexp
pattern=(SEC_STARTUP|SEC_RESTART)
context=SEC_INTERNAL_EVENT
desc=Init counters with 0
action=lcall %o ->( sub { %ssh_summary=(); } )
# note that if exec was used instead of lcall,
# the prior line would need to escape the % and would be:
# action=exec %o ( %%ssh_summary=(); )
```

Then add a rule to output the stats daily and reset them.

```
type=calendar
time=0 0 * * *
desc=output daily stats
action=lcall %o -> ( sub { $ssh_summary{total_sessions}; } ); \
  udgram /dev/log <30>sec-summary: There were %o ssh sessions
today; \
  lcall %n -> ( sub { my($ret); \
        foreach (keys %{$ssh_summary{user}}) { \
           $ret .= "$_ = $ssh_summary{user}{$_}{count} "; } \
        $ssh_summary{total_sessions} = 0; return $ret; } ); \
  if %n ( udgram /dev/log <30>sec-summary: Number of SSH \
sessions for each user: %n )
```

Another good use of Perl is to load a table at startup, and then test it during the rules.

For example, if you create a file that contains a list of your production server names, and then create a startup rule like:

```
type=single
ptype=regexp
pattern=(SEC_STARTUP|SEC_RESTART)
context=SEC_INTERNAL_EVENT
desc=Load Production Server Table
action=eval %o (%%prodservers=();open(infile, \
"</etc/prodservers.txt"); \
    while <infile> {chomp; $prodservers{$_}=1; }; close(infile))
```

you can then add a context test to our original example rule to only alert if the log is from a production server.

```
context= =(exists $prodservers{$1})
```

Similar to the exec command, this compiles the code on each run (and requires escaping % characters). There is the equivalent to lcall that would look like:

```
context= $1 -> ( sub { exists $prodservers{$_[0]} } )
```

You can have SEC reload the table on demand by adding a rule like:

```
type=single
ptype=regexp
pattern=SEC reload production server table
desc=Reload Production Server Table
action=eval %o (%%prodservers=();open(infile, \
"</etc/prodservers.txt"); \
    while <infile> {chomp; $prodservers{$_}=1; }; close(infile))
```

Note that you probably want to have additional restrictions so that the reload can only be generated by logs from a specific set of servers.

Caching Match Results

When you have a number of tests that you want to do with a single log event, doing the same regexp repeatedly is inefficient.

Using our example, let's say you want to do multiple alerts on ssh logins. Instead of each of the rules needing to rerun the regexp against the log line, you could add the following line to the original example rule:

```
varmap=ssh; line=0; server=1; user=2;source =3; port=4
```

Then you could create a second rule such as:

```
type=singlewiththreshold
ptype=cached
pattern=ssh
desc=lots of logins for user $+{user}
action=write - $+{user} logged in to more than 5 servers in one
minute
window=60
thresh=5
continue=takenext
```

With the use of Perl in the context, you could configure this to only fire if the user has logged in to more than ten servers all day (to avoid getting alerts when users arrive in the morning and log in to a bunch of places to start their day) by adding a line:

```
context=ssh :> (sub { \
    return $ssh_summary{user}{$_[0]->{user}}{count} > 10} )
```

Debugging

Debugging alerting systems is always an interesting exercise. You need to be able to generate events to trigger the rules, but when they don't fire as expected, you need to be able to figure out what the internal state of your alerting engine is. SEC provides this option by way of dump files. If you start SEC with the option `--dump=/path/to/dumpfile`, you can send it the signal USR1, and if the dump file does not already exist, SEC will dump its internal state. This includes performance stats, how many matches there have been for each rule, the last several lines that it has processed, and information about every context that it is tracking.

Another approach to debugging is to run SEC from the command line with it reading from stdin or a file. SEC generates a lot of output as it processes each message, telling you what it does each step of the way; however, the types of problems that are the hardest to troubleshoot tend to involve timing, which means that you can't just use a test file. The timing in SEC is based on when SEC sees the log entry, not any timestamp that may appear in the log entry. You are better off generating the input to SEC with a script so that you have a repeatable test that generates the correct messages with the right timing, either echo+sleep or logger+sleep if you want to test any filtering in rsyslog as well.

Conclusion

This is a brief overview of the capabilities of SEC, and there are a lot of nuances and other capabilities that I did not go into. With the different test types, contexts, desc fields, alerting scripts, and embedded Perl snippets, there is little that SEC cannot do.

SEC does take some training and expertise to master and configure for your environment, but any serious alerting engine that you use is going to require customization to your needs. The biggest problem with SEC is that there is not a good pool of examples available for people to work from, but the users mailing list [4] is extremely responsive to requests.

References

[1] David Lang, Enterprise Logging: https://www.usenix.org/publications/login/august-2013-volume-38-number-4/enterprise-logging and https://www.usenix.org/publications/login/october-2013-volume-38-number-5/log-filtering-rsyslog.

[2] http://simple-evcorr.sourceforge.net/.

[3] http://en.wikipedia.org/wiki/Syslog.

[4] https://lists.sourceforge.net/lists/listinfo/simple-evcorr-users/.

Erasure Codes for Storage Systems
A Brief Primer

JAMES S. PLANK

James S. Plank received his BS from Yale University in 1988 and his PhD from Princeton University in 1993. He is a professor in the Electrical Engineering and Computer Science Department at the University of Tennessee, where he has been since 1993. His research interests are in fault-tolerance, erasure coding, storage systems, and distributed computing. He is a past Associate Editor of IEEE Transactions on Parallel and Distributed Computing, and a member of the IEEE Computer Society. plank@cs.utk.edu

Storage systems have grown to the point where failures are inevitable, and those who design systems must plan ahead so that precious data is not lost when failures occur. The core technology for protecting data from failures is erasure coding, which has a rich 50+ year history stemming communication systems, and as such, can be confusing to the storage systems community. In this article, I present a primer on erasure coding as it applies to storage systems, and I summarize some recent research on erasure coding.

Storage systems come in all shapes and sizes, but one thing that they all have in common is that components fail, and when a component fails, the storage system is doing the one thing it is not supposed to do: losing data. Failures are varied, from disk sectors becoming silently corrupted, to entire disks or storage sites becoming unusable. The storage components themselves are protected from certain types of failures. For example, disk sectors are embedded with extra-correcting information so that a few flipped bits may be tolerated; however, when too many bits are flipped, or when physical components fail, the storage system sees this as an erasure: the storage is gone!

To deal with these failures, storage systems rely on erasure codes. An erasure code adds redundancy to the system to tolerate failures. The simplest of these is replication, such as RAID-1, where each byte of data is stored on two disks. In that way any failure scenario may be tolerated, so long as every piece of data has one surviving copy. Replication is conceptually simple; however, it consumes quite a lot of resources. In particular, the storage costs are doubled, and there are scenarios in which two failed storage components (those holding both copies of a piece of data) lead to data loss.

More complex erasure codes, such as the well-known Reed-Solomon codes, tolerate broader classes of failure scenarios with less extra storage. As such, they are applicable to today's storage systems, providing higher levels of fault-tolerance with less cost. Unfortunately, the field of erasure coding traces its lineage to error correcting codes (ECC) in communication systems, where they are used to solve a similar-sounding but in reality quite different problem. In communications, errors arise when bits are corrupted silently in a message. This differs from an erasure, because the location of the corruption is unknown. The fact that erasures expose the location of the failure allows for erasure codes to be more powerful than ECCs; however, classic treatments of erasure codes present them as special cases of ECCs, and their application to storage systems is hard to glean.

In this article, I explain erasure codes in general as they apply to storage systems. I will first present nomenclature and general erasure coding mechanics, and then outline some common erasure codes. I then detail some of the more recent research results concerning erasure codes and storage systems. I provide an annotated bibliography at the end of this article so that the interested reader may explore further.

The Mechanics of Simple Codes

Let's assume that our storage system is composed of n disks. We partition them into k disks that hold user data so that m=n−k disks hold coding information. I refer to them as data and

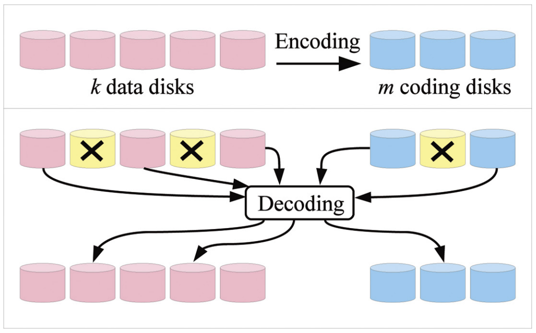


Figure 1: An erasure-coded storage system encodes k data disks onto m coding disks. When up to m disks fail, their contents are decoded by the erasure code.

coding disks, respectively. The acts of encoding and decoding are pictured in Figure 1.

With encoding, the contents of the k data disks are used to calculate the contents of the m coding disks. When up to m disks fail, their contents are decoded from the surviving disks. Repeating from above, when a disk fails, the failure mode is an erasure, where its contents are considered to be unreadable.

The simplest erasure codes assume that each disk holds one w-bit word. I label these words $d_0, \ldots, d_{k-1}$, which are the data words stored on the data disks, and $c_0, \ldots, c_{m-1}$, which are the coding words stored on the coding disks. The coding words are defined as linear combinations of the data words:

$$c_0 = a_{(0,0)}d_0 + \ldots + a_{(0,k-1)}d_{k-1}$$
$$c1 = a_{(1,0)}d_0 + \ldots + a_{(1,k-1)}d_{k-1}$$
$$\ldots$$
$$\ldots$$
$$c_{m-1} = a_{(m-1,0)}d_0 + \ldots + a_{(m-1,k-1)}d_{k-1}$$

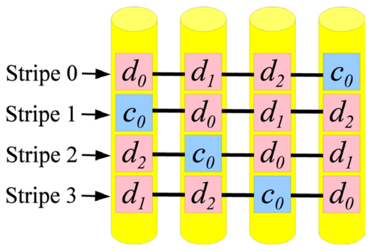
The coefficients a are also w-bit words. Encoding, therefore, simply requires multiplying and adding words, and decoding involves solving a set of linear equations with Gaussian elimination or matrix inversion.

The arithmetic of erasure coding is special. When w=1, all of the d, c and a variables are single bits, and the arithmetic is standard arithmetic modulo 2: addition is binary XOR (⊕) and multiplication is binary AND. When w is larger, the arithmetic is called Galois Field arithmetic, denoted $GF(2^w)$. This arithmetic operates on a closed set of numbers from 0 to 2^{w-1} in such a way that addition, multiplication, and division all have the properties that we expect. Conveniently, addition in a Galois Field is equal to bitwise XOR. Multiplication is more complicated, and beyond the scope of this article; however, there is a great deal of reference material on Galois Field arithmetic plus a variety of open source implementations (please see the annotated bibliography).

A disk, of course, holds more than a single w-bit word; however, with these simple codes, I partition each disk into w-bit words, and the i-th words on each disk are encoded and decoded together, independently of the other words. So that disks may be partitioned evenly into w-bit words, w is typically selected to be a power of two. Popular values are w=1 for its simplicity, because the arithmetic is composed of XORs and ANDs, and w=8, because each word is a single byte. In general, larger values of w allow for richer erasure codes, but the Galois Field arithmetic is more complex computationally.

An erasure code is therefore defined by w and the coefficients $a_{(i,j)}$. If the code successfully tolerates the failures of any m of the n disks, then the code is optimal with respect to fault-tolerance for the amount of extra space dedicated to coding. This makes sense, because one wouldn't expect to add m disks of redundancy and be able to tolerate more than m disk failures. If a code achieves this property, it is called *maximum distance separable* (MDS), a moniker that conveys zero intuition in a storage system. Regardless, MDS codes are desirable, because they deliver optimal fault tolerance for the space dedicated to coding.

In real storage settings, disks are partitioned into larger units called strips, and the set of corresponding strips from each of the n disks that encode and decode together is called a *stripe*. Each stripe is an independent entity for erasure coding, which allows the storage system designer to be flexible for a variety of reasons. For example, one may wish to rotate the identities of the n disks on a stripe-by-stripe basis, as in the left side of Figure 2. This is



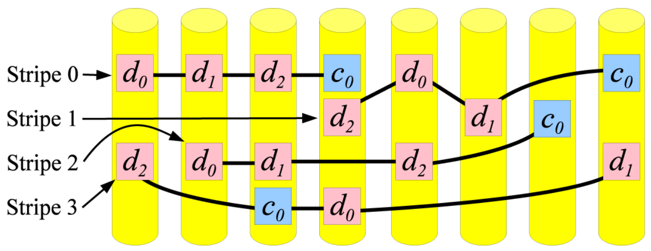


Figure 2: Two examples of laying out stripes on a collection of disks. On the left, there are n=4 disks, and each stripe contains k=3 strips of data and m=1 of coding. So that load is balanced, each stripe rotates the identities of the disks. On the right, there are now eight disks; however stripes still contain n=4 strips, three of which are data and one of which is coding.

a balanced approach, where each of the n=4 disks contains the same ratio of data and coding strips.

On the right side, a more ad hoc approach to laying out stripes is displayed. There are eight disks in this system; however, each stripe is composed of three data strips and one coding strip. Thus, the erasure code may be the same as in the left side of the figure; the allocation of strips to stripes is the only difference. This approach was used by Panasas to allow for flexible block allocation, and to allow additional disks to be added seamlessly to the storage system.

RAID-4 and RAID-5

Within this framework, I can define RAID-4 and RAID-5 as using the same simple erasure code, but having different stripe layouts. The code is an MDS code where m=1, w=1, and all of the a coefficients are also one. The sole coding bit is labeled p, and it is the XOR of all of the data bits:

$$p = d_0 \oplus d_1 \oplus \ldots \oplus d_{k-1}.$$

When any bit is erased, it may be decoded as the XOR of the surviving bits.

Although this equation operates on single bits, its implementation in a real system is extremely efficient, because whole strips may be XOR'd together in units of 128 or 256 bits using vector instructions such as Intel SSE2 (128 bits) or AVX (256 bits).

RAID-4 and RAID-5 both use the same erasure code; however, with RAID-4, the identity of each disk is fixed, and there is one disk, P, dedicated solely to coding. With RAID-5, the identities are rotated on a stripe-by-stripe basis as in the left side of Figure 2. Therefore, the system is more balanced, with each disk equally holding data and coding.

Linux RAID-6

RAID-6 systems add a second disk (called Q) to a RAID-4/5 system and tolerate the failure of any two disks. This requires an MDS erasure code where m=2, which is impossible to achieve with a simple XOR code. The solution implemented by the Red Hat Linux kernel employs the following simple code for w=8:

$$p = d_0 \oplus d_1 \oplus \ldots \oplus d_{k-1}$$
$$q = d_0 \oplus 2(d_1) \oplus \ldots \oplus 2^{k-1}(d_{k-1})$$

This code has some interesting properties. First, because addition in a Galois Field is equivalent to XOR, the P disk's erasure coding is equivalent to RAID-4/5. Second, the Q disk may be calculated using only addition and multiplication by two, because:

$$q = 2\ (\ 2\ (\ \ldots 2\ (2d_{k-1} \oplus d_{k-2})\ \ldots) \oplus d_1\ ) \oplus d0.$$

This is important because there are techniques to multiply 128- and 256-bit vectors of bytes by two in GF(28) with a small number of SSE/AVX instructions.

Reed-Solomon Codes

Reed-Solomon codes are MDS codes that exist whenever $n \le 2^w$. For example, so long as a storage system contains 256 disks or less, there is a Reed-Solomon defined for it that uses arithmetic in GF(2^8). There are multiple ways to define the $a_{(i,j)}$ coefficients. The simplest to explain is the "Cauchy" construction: Choose n distinct numbers in GF(2^w) and partition them into two sets X and Y such that X has m elements and Y has k. Then:

$$a_{(ij)} = \frac{1}{x_i \oplus y_j} ,$$

where arithmetic is over GF(2^w).

Reed-Solomon codes are important because of their generality: they exist and are easy to define for any value of k and m. They have been viewed historically as expensive, because the CPU complexity of multiplication in a Galois Field is more expensive than XOR; however, vector instruction sets such as Intel SSE3 include operations that enable one to multiply 128-bit vectors of bytes by constants in a Galois Field with a small number of instructions. Although not as fast as multiplying by two as they do for a RAID-6 Q disk, it is fast enough that in most Reed-Solomon coding installations, disk I/O and even cache speeds are larger bottlenecks than the CPU. There are multiple open source libraries that implement Reed-Solomon coding for storage installations.

Array Codes

Array codes for storage systems arose in the 1990s. They were motivated by the desire to avoid Galois Field arithmetic and implement codes solely with the XOR operation. In the simple codes above, each disk logically holds one w-bit word, and thus there are m coding words, each of which is a different linear combination of the k data words. In an array code, each disk holds r w-bit words. Thus, there are mr coding words, each of which is a different linear combination of the kr data words.

They are called "array codes" because the coding system may be viewed as an r × n array of words, where the columns of the array are words that are co-located on the same disk. I depict an example in Figure 3. This is the RDP erasure code for k=4 and m=2. As such, it is a RAID-6 code. Each disk holds four bits, which means that r=4 and w=1. In the picture, I draw the array with the Q words on the left, the P words on the right, and the data words in the middle. The horizontal gray bars indicate XOR equations for the P disk's bits, and the other lines indicate how the Q disk's bits are encoded.

The allure of array codes for w=1 is that encoding and decoding require only XOR operations, yet the codes may be defined so that they are MDS. Examples are RDP, EVENODD, Blaum-Roth and Liberation codes for RAID-6, the STAR code for m=3, and Cauchy Reed-Solomon, Generalized EVENODD and Generalized RDP, which are defined for all values of k and m.

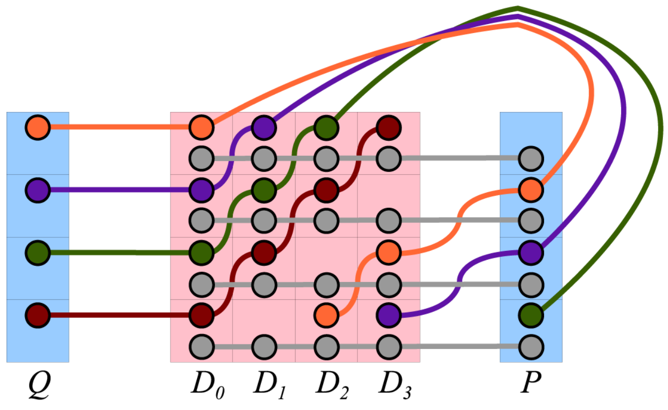


Figure 3: The RDP array code with the following parameters: k=4, m=2 (RAID-6), n = k+m = 6, r=4, w=1. The gray lines depict the coding equations for the P disk. The other lines depict the coding equations for the Q disk.

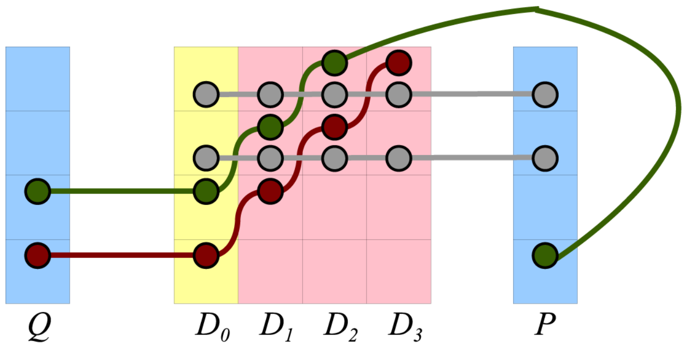


Figure 4: Recovering from a single failure in RDP. Only 12 bits are required, as opposed to 16 bits when one recovers solely from the P disk.

As mentioned in the section on mechanics, above, the advent of vector instructions has lowered the CPU burden of Galois Field arithmetic, and thus the allure of array codes has diminished in recent years; however, they have interesting properties with respect to recovery that make them viable alternatives to the simple codes. I explain this in the next section.

Recent Work #1: Reduced Disk I/O For Recovery

When one is using a simple erasure code and a single disk fails, the only way to recover its contents is to pick one of the m coding equations and use it to decode. This requires one to read k–1 strips from the surviving disks to calculate each strip on the failed disk. With an array code, one may significantly reduce the amount of data that is read from the surviving disks. I present an example in Figure 4, using the RDP code from Figure 3. In this example, the disk D_0 has failed and needs to be decoded. Were a simple erasure code employed, recovery would be equivalent to decoding solely from the P drive, where 16 bits must be read from the surviving disks; however, because of the structure of RDP, a judicious choice of decoding equations from both the P and Q drive allows one to decode D_0 by reading only 12 bits from the surviving disks.

As described in the section on mechanics, each bit in the description of the code corresponds to a larger block of storage on disk, which means that this example reduces the I/O costs of recovery by 25 percent. This observation was first made by Xiang in 2010, and further research has applied it to other array codes.

Recent Work #2: Regenerating Codes

Regenerating codes focus on reducing network I/O for recovery in distributed, erasure-coded storage systems. When one or more storage nodes fail, the system replaces them, either with nodes that hold their previous contents, or with nodes that hold equivalent contents from an erasure-coding perspective. In other words, the new collection of n nodes may hold different contents than the old collection; however, it maintains the property that the data may be calculated from any k of the nodes.

The calculation of these new nodes is performed so that network I/O is minimized. For example, suppose one storage node has failed and must be replaced. A simple erasure code requires k–1 of the other nodes to read their contents and send them for reconstruction. The schemes from Xiang (previous section) may leverage array codes so that more than k–1 nodes read and transmit data, but the total amount of data read and transmitted is reduced from the simple case. A properly defined regenerating code has the surviving nodes read even more data from disk, but then they massage it computationally so that they transmit even less data to perform regeneration.

Research on regenerating codes is both active and prolific. Please see the bibliography for summaries and examples.

Recent Work #3: Non-MDS Codes

A non-MDS code does not tolerate all combinations of m failures, and therefore the fault-tolerance is not optimal for the amount of extra storage committed to erasure coding; however, relaxation of the MDS property is typically accompanied by performance improvements that are impossible to achieve with MDS codes. I give a few examples here.

Flat XOR codes are simple codes where w=1. When m > 1, they are non-MDS; however, they have attractive features in comparison to their MDS counterparts. First, since w=1, they are based solely on the XOR operation—no Galois Field arithmetic is required. Second, they reduce both the I/O and the CPU complexity of encoding and decoding. When k and m grow to be very large (in the hundreds or thousands), flat XOR codes like Tornado and Raptor codes provide good degrees of fault-tolerance, while only requiring small, constant numbers of I/Os and XORs

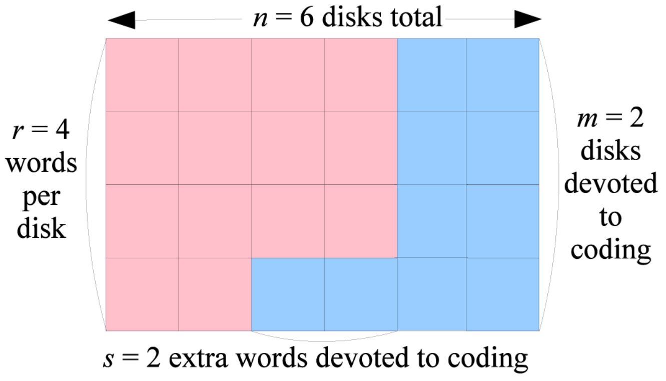

$n = 6$ disks total
$r = 4$ words per disk
$m = 2$ disks devoted to coding
$s = 2$ extra words devoted to coding

Figure 5: The layout of a stripe with an SD code, which tolerates the failure of any two disks and any additional two words in the stripe.

for encoding and decoding. This is as opposed to an MDS code, which necessarily requires O(k) I/Os and arithmetic operations. Other non-MDS codes that reduce complexity and rely solely on XOR are HoVeR, WEAVER, and GRID.

A second important class of non-MDS codes partitions the data words into groups, and divides the coding words into "local parities" and "global parities." Each local parity word protects a single group of data words, whereas each global parity word protects all of the words. The system is then fault-tolerant to a certain number of failures per data group, plus an additional number of failures for the entire system. The computational and I/O costs are smaller than an MDS system, yet the failure coverage is provably optimal for this coding paradigm. Examples of these codes are LRC codes that are implemented in Microsoft's Azure storage system, an identically named but different LRC code that has an open-source implementation in Hadoop, and Partial-MDS codes from IBM.

Finally, Sector-Disk (SD) codes are a class of non-MDS codes where m disks and s sectors per stripe are dedicated to fault-tolerance. An example is drawn in Figure 5, where a 6-disk system requires each disk to hold four words in its stripe. Two disks are devoted to fault-tolerance, and two additional words in the stripe are also devoted to fault-tolerance. The codes are designed so that they tolerate the failure of any two disks and any two additional words in the stripe. Thus, their storage overhead and fault-tolerance match the mixed failure modes of today's disks, where sector failures accumulate over time, unnoticed until a disk failure requires that they be read for recovery.

Conclusion

In this article, I have presented how erasure codes are leveraged by storage systems to tolerate the failure of disks, and in some cases, parts of disks. There are simple erasure codes, such as RAID-4/5 and Reed-Solomon codes, that view each disk as holding a single w-bit word, and define the coding words as linear combinations of the data words, using either XOR or Galois Field arithmetic. Array codes view each disk as holding multiple w-bit words, and achieve richer fault-tolerance, especially for codes based solely on the XOR operation. More recent work has focused on reducing the disk and network I/O requirements of the erasure codes, and on loosening the fault-tolerance requirements of the codes to improve performance.

Annotated Bibliography

In this section, I provide reference material for the various topics in the article. The following papers provide reference on implementing Galois Field arithmetic for erasure coding, including how to use vector instructions to accelerate performance drastically. The paper by Anvin [5] details the Linux RAID-6 implementation of Reed-Solomon coding.

[1] K. Greenan, E. Miller, and T. J. Schwartz. Optimizing Galois Field arithmetic for diverse processor architectures and applications. In MASCOTS 2008: 16th IEEE Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Baltimore, MD, September 2008.

[2] J. Luo, K. D. Bowers, A. Oprea, and L. Xu. Efficient software implementations of large finite fields GF(2n) for secure storage applications. *ACM Transactions on Storage* 8(2), February 2012.

[3] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software—Practice & Experience* 27(9):995–1012, September 1997.

[4] J. S. Plank, K. M. Greenan, and E. L. Miller. Screaming fast Galois Field arithmetic using Intel SIMD instructions. In FAST-2013: 11th USENIX Conference on File and Storage Technologies, San Jose, February 2013.

[5] H. P. Anvin. The mathematics of RAID-6: http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf, 2009.

[6] H. Li and Q. Huan-yan. Parallelized network coding with SIMD instruction sets. *International Symposium on Computer Science and Computational Technology,* IEEE, December 2008, pp. 364–369.

The following are open-source implementations of Galois Field arithmetic and erasure coding:

[7] Onion Networks. Java FEC Library v1.0.3. Open source code distribution: http://onionnetworks.com/fec/javadoc/, 2001.

[8] A. Partow. Schifra Reed-Solomon ECC Library. Open source code distribution: http://www.schifra.com/downloads.html, 2000-2007.

[9] J. S. Plank, K. M. Greenan, E. L. Miller, and W. B. Houston. GF-Complete: A comprehensive open source library for Galois

Field arithmetic. Technical Report UT-CS-13-703, University of Tennessee, January 2013.

[10] J. S. Plank, S. Simmerman, and C. D. Schuman. Jerasure: A library in C/C++ facilitating erasure coding for storage applications—Version 1.2. Technical Report CS-08-627, University of Tennessee, August 2008.

[11] L. Rizzo. Erasure codes based on Vandermonde matrices. Gzipped tar file posted: http://planete-bcast.inrialpes.fr/rubrique.php3?id_rubrique=10, 1998.

Besides my tutorial on Reed-Solomon coding for storage systems [3], the textbook by Peterson describes Reed-Solomon coding in a more classic manner. The papers by Blomer et al. and Rabin explain the "Cauchy" Reed-Solomon coding construction:

[12] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman. An XOR-based erasure-resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, August 1995.

[13] W. W. Peterson and E. J. Weldon, Jr. *Error-Correcting Codes, Second Edition.* The MIT Press, Cambridge, Massachusetts, 1972.

[14] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM* 36(2):335–348, April 1989.

The following papers describe array codes for RAID-6 that are based solely on the XOR operation:

[15] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computing* 44(2):192–202, February 1995.

[16] M. Blaum and R. M. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory* 45(1):46–59, January 1999.

[17] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row diagonal parity for double disk failure correction. In FAST-2004: 3rd USENIX Conference on File and Storage Technologies, San Francisco, CA, March 2004.

[18] J. S. Plank, A. L. Buchsbaum, and B. T. Vander Zanden. Minimum density RAID-6 codes. *ACM Transactions on Storage* 6(4), May 2011.

Blomer et al.'s paper [12] describes how to convert a standard Reed-Solomon code into an array code that only uses XORs. The next three papers describe other general MDS array codes where w=1:

[19] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *IEEE Transactions on Information Theory* 42(2):529—542, February 1996.

[20] M. Blaum. A family of MDS array codes with minimal number of encoding operations. In IEEE International Symposium on Information Theory, Seattle, September 2006.

[21] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers* 57(7):889–901, July 2008.

The following papers reduce the amount of data that must be read from disk when performing recovery on XOR-based array codes:

[22] O. Khan, R. Burns, J. S. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. In FAST-2012: 10th USENIX Conference on File and Storage Technologies, San Jose, February 2012.

[23] Z. Wang, A. G. Dimakis, and J. Bruck. Rebuilding for array codes in distributed storage systems. In *GLOBECOM ACTEMT Workshop,* pp. 1905–1909. IEEE, December 2010.

[24] L. Xiang, Y. Xu, J. C. S. Lui, and Q. Chang. Optimal recovery of single disk failure in RDP code storage systems. In ACM SIGMETRICS, June 2010.

The following papers summarize and exemplify research on regenerating codes:

[25] V. Cadambe, C. Huang, J. Li, and S. Mehrotra. Compound codes for optimal repair in MDS code based distributed storage systems. In Asilomar Conference on Signals, Systems and Computers, 2011.

[26] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory,* 2010.

[27] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE* 99(3), March 2011.

The following papers describe XOR-based, non-MDS codes that improve the performance of encoding and recovery:

[28] K. M. Greenan, X. Li, and J. J. Wylie. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery and tradeoffs. In 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST2010), Nevada, May 2010.

[29] J. L. Hafner. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. In *FAST-2005: 4th USENIX Conference on File and Storage Technologies,* pp. 211–224, San Francisco, December 2005.

[30] J. L. Hafner. HoVer erasure codes for disk arrays. In DSN-2006: The International Conference on Dependable Systems and Networks, Philadelphia, June 2006.

[31] M. Li, J. Shu, and W. Zheng. GRID codes: Strip-based erasure codes with high fault tolerance for storage systems. *ACM Transactions on Storage* 4(4), January 2009.

[32] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *29th Annual ACM Symposium on Theory of Computing,* pages 150–159, El Paso, TX, 1997. ACM.

[33] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory,* pages 2551–2567, 2006.

The following papers describe non-MDS erasure codes that feature local and global parity words and address cloud storage systems or mixed failure modes in RAID systems:

[34] M. Blaum, J. L. Hafner, and S. Hetzler. Partial-MDS codes and their application to RAID type of architectures. *IEEE Transactions on Information Theory,* July 2013.

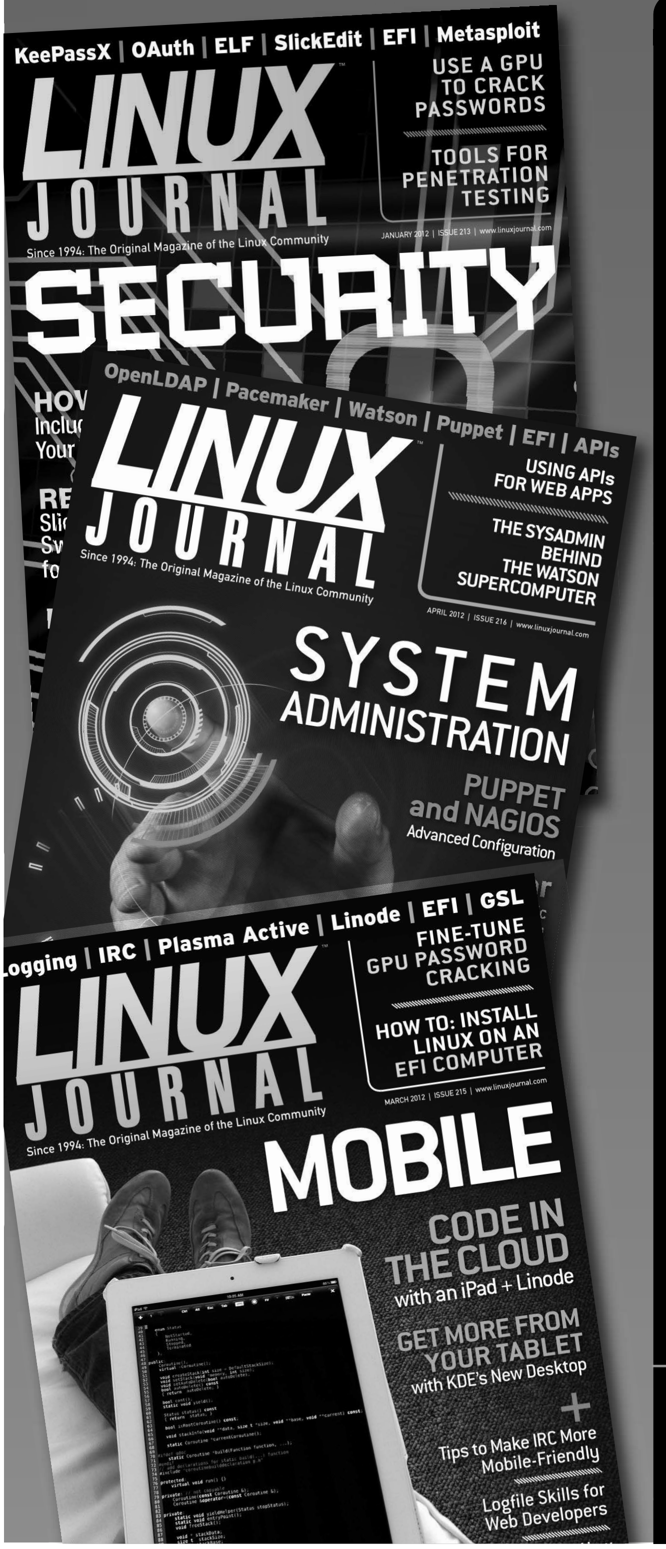
[35] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in Windows Azure storage. In USENIX Annual Technical Conference, Boston, June 2012.

[36] J. S. Plank, M. Blaum, and J. L. Hafner. SD codes: Erasure codes designed for how storage systems really fail. In FAST-2013: 11th USENIX Conference on File and Storage Technologies, San Jose, February 2013.

[37] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing elephants: Novel erasure codes for big data. In 39th International Conference on Very Large Data Bases, August 2013.

Acknowledgements
The author thanks Rik Farrow for encouraging him to write this article. This material is based upon work supported by the National Science Foundation under grant CSR-1016636, and by an IBM Faculty Research Award. The author is indebted to his various erasure-coding cohorts and co-authors through the years: Mario Blaum, Randal Burns, Kevin Greenan, Jim Hafner, Cheng Huang, Ethan Miller, Jason Resch, Jay Wylie, and Lihao Xu.

If You Use Linux, You Should Be Reading LINUX JOURNAL™



KeePassX | OAuth | ELF | SlickEdit | EFI | Metasploit
LINUX JOURNAL
USE A GPU TO CRACK PASSWORDS
TOOLS FOR PENETRATION TESTING
SECURITY
OpenLDAP | Pacemaker | Watson | Puppet | EFI | APIs
USING APIs FOR WEB APPS
THE SYSADMIN BEHIND THE WATSON SUPERCOMPUTER
SYSTEM ADMINISTRATION
PUPPET and NAGIOS Advanced Configuration
Logging | IRC | Plasma Active | Linode | EFI | GSL
FINE-TUNE GPU PASSWORD CRACKING
HOW TO: INSTALL LINUX ON AN EFI COMPUTER
MOBILE
CODE IN THE CLOUD with an iPad + Linode
GET MORE FROM YOUR TABLET with KDE's New Desktop
Tips to Make IRC More Mobile-Friendly
Logfile Skills for Web Developers

» In-depth information providing a full 360-degree look at featured topics relating to Linux

» Tools, tips and tricks you will use today as well as relevant information for the future

» Advice and inspiration for getting the most out of your Linux system

» Instructional how-tos will save you time and money

Subscribe now for instant access! For only $29.50 per year—less than $2.50 per issue—you'll have access to *Linux Journal* each month as a PDF, in ePub & Kindle formats, on-line and through our Android & iOS apps. Wherever you go, *Linux Journal* goes with you.

SUBSCRIBE NOW AT:
WWW.LINUXJOURNAL.COM/SUBSCRIBE

COLUMNS

Practical Perl Tools

CLI Me a River

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 28+ 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 2005 conference and one of the LISA 2006 Invited Talks co-chairs. David is honored to be the recipient of the 2009 SAGE Outstanding Achievement award and to serve on the USENIX Board of Directors.
dnb@ccs.neu.edu

If Neil Stephenson's 1999 60+ page essay "In the Beginning was the Command Line" [1] left you feeling "ooh, a famous science fiction author really gets me, he really understands what is in my heart," then this column is for you. Today we're going to talk about a few ways you can write a better command-line program in Perl.

Switch It Up

One of the first things that contributes to someone's aesthetic pleasure of using a command-line tool is how well it handles arguments/switches. There are at least two sets of choices at work here. The first is a design one that Perl isn't going to help one whit with. Coming up with switch names that make sense for your program, are the same as or like the names used in similar programs in the same domain, are clear, and so on is up to you. This is by no means an easy task, because it requires careful thought.

The second set of choices does have a technical solution. The second set of choices is the one where you decide how your program will accept the arguments. Will there be spaces between them? Can you abbreviate and/or combine switches? Are some mandatory? And so on . . . This all matters because you want, whenever possible, for someone to try the arguments using the first way that comes into her head and have it work.

Where Perl helps with this is there are modules (oh so many modules) that handle argument-parsing for you. A number of them will handle all of the fiddly details for you so that your program can be liberal in how the arguments are specified (one dash, two dashes, abbreviated, abbreviated to single letters, optional and required arguments, and so on). The variety is dizzying. Before I show you one of these modules, I should mention that the Perl interpreter actually implements a built-in argument processor in the form of the -s switch. This means you can write code that looks like this:

```
#!/usr/bin/perl -s

if ($add)    { print "You want to add $add\n"; }
if ($remove) { print "You want to remove $remove\n"; }


if (${-help}) { print "this variable is crazy!\n"; }
```

which, when run, gives you:

```
$ s.pl -add=fred
You want to add fred
$ s.pl -remove
You want to remove 1
$ s.pl --help
this variable is crazy!
```

But don't write code that looks like that. The -s switch takes anything passed in with a dash, strips off the dash, and puts it in a variable with the name of the argument. This has all sorts of fun ramifications, a couple of which are mentioned in the perlrun doc:

> Do note that a switch like --help creates the variable "${-help}", which is not compliant with "use strict 'refs'". Also, when using this option on a script with warnings enabled you may get a lot of spurious "used only once" warnings.

In short, this means that you can kiss "use strict;", the thing everyone tells you to put first in our programs, goodbye unless you are willing to turn off some of the strictness.

Out of the crazy number of command argument parsing modules out there I'm only going to pick one to demonstrate. This is clearly a subject Perl authors like to riff on, so if it doesn't float your boat I'd encourage you to spend some time searching CPAN for one that does. And if you are a budding Perl module author who has aspirations of writing your own command argument-parsing module, I'd beseech you to check CPAN multiple times for something that works for you before reinventing yet another wheel.

The module we're going to explore is one of the most popular modules in this space, perhaps because it actually ships with Perl. Let's take a quick look at Getopt::Long. Getopt::Long can do so many things that the long manual page might be a bit daunting on first glance. We'll start with its sample code and then spice things up as we go along:

```
use Getopt::Long;
my $data   = "file.dat";
my $length = 24;
my $verbose;
GetOptions ("length=i" => \$length,    # numeric
            "file=s"   => \$data,      # string
            "verbose"  => \$verbose)   # flag
or die("Error in command line arguments\n");
```

The key function here is the GetOptions() call. The variable assignments before it are both to keep a "use strict" line (omitted in the sample code for space reasons) happy and probably just to reaffirm what kind of data is being referenced in the GetOptions call. Let's take that call apart.

In general, GetOptions takes a hash that defines the name of an argument, what kind of value it must or can be set to (numeric, string, etc.), any special characteristics (like "required" or "optional"), and a reference to a place to put the information parsed from the command-line arguments. For example, this part:

```
"length=i" => \$length,
```

says if we get an argument called length (--length), it must take a value and that value has to be an integer. That value will be stored in $length (i.e., --length 2 will put '2' in $length). In the case of a flag (like --verbose), the variable gets set to "1" so that Boolean tests like "if ($verbose)" will act as expected.

Two quick things to note before we start to add to this example code. The "or die()" that follows GetOptions works because GetOptions returns true if it can parse the options according to your wishes, false if that failed (e.g., someone passed in an argument you hadn't specified). The other thing to note is Getopt::Long by default will let you abbreviate unambiguous arguments on the command line and will handle multiple formats. This means I could call the program with:

```
$ s2.pl --length 2
$ s2.pl --length=2
$ s2.pl --l 2
$ s.2pl --l=2
$ s.2pl --le 2
```

and so on. Note that I don't have to code anything special to handle all of these different variations. This is what I mean by having Perl make it easier to make better command-line programs.

A moment ago, I said we could add to the sample code, so let me give you a list of how we can make the argument processing even fancier:

- Optional values (using : instead of = as in length:i)
- Multiple values per flag (pass a reference to an array instead of a scalar)
- Negated flags (i.e., --noverbose, which then sets $verbose to 0 instead of 1, specified by using an exclamation mark after the argument name)
- Cumulative flags (i.e., -v -v -v will give you more verbose output, specified by using + after the argument name)
- Argument name aliases (use different names for the same argument, specified by using a pipe character in the name, as in "verbose|chatty|moar" => \$verbose)

Getopt::Long has a few other tricks up its sleeve that I encourage you to go read about. The only one I want to mention before we move on is one I use on a regular basis. I haven't been very explicit about this, but hopefully you've sussed out that the way the rest of your program can determine which arguments and values were specified on the command line is through the variables being set by GetOptions(). I prefer to be able to find all of my options in a single place vs. a bunch of unconnected variables. To do that, we can tell GetOptions to store everything in a single hash by providing a reference to that hash as the first argument like so:

```
my %options = ();
GetOptions(\%options,
        "length=i"  => \$length,
        "file=s"    => \$data,
        "verbose"   => \$verbose);
```

When you do it that way, you can reference $options{length}, $options{file} and $options{verbose}. To check to see if an option has been set, you'll want to do something like

```
  if ( exists $options{verbose} ) { ... }
```

As I mentioned before, there are tons of variations on the argument-parsing theme. Some of the variations I found most compelling are those that construct the argument specification from a script's internal documentation (e.g., in POD form). This leads nicely into the next topic.

Do the Doc

In the previous section I brought up the notion that we are endeavoring to design things like switch names to be intuitive and sensical to the script's users. But even if you manage to intuit or sense the heck out of your users (if that is even a term), there are still going to be times where those users will want to see a list of possible arguments and, ideally, some documentation for them.

That's where the module Pod::Usage comes into play. We've talked about this module back in 2006 and earlier this year, but I still want to remind you about it because having a mechanism for providing this documentation is pretty key to a good command-line program. You'll forgive me if I do as I did in one of those columns and reproduce the sample code from the Pod::Usage documentation, because it really does offer the best example for how to use the module. Plus, it even uses Getopt::Long, tying nicely into the last subject. Here's the sample code minus the actual specification of the USAGE and manual page in POD form:

```
    use Getopt::Long;
    use Pod::Usage;

    my $man = 0;
    my $help = 0;
    ## Parse options and print usage if there is a syntax error,
    ## or if usage was explicitly requested.
    GetOptions('help|?' => \$help, man => \$man) or pod2usage(2);
    pod2usage(1) if $help;
    pod2usage(-verbose => 2) if $man;

    ## If no arguments were given, then allow STDIN to be used only
    ## if it's not connected to a terminal (otherwise print usage)
    pod2usage("$0: No files given.")  if ((@ARGV == 0) && (-t STDIN));
    __END__
```

Okay, so let's see what is going on here. Our newfound friend, GetOptions() from Getopt::Long, is being called to look for either an argument called "help" or "man". When it gets one of those two arguments, it calls pod2usage() with a return code and/or a "verbosity" level. A verbosity level of 0 shows an abridged USAGE message: 1 spits out the full USAGE message and 2 will print out the entire man page. Pod::Usage has rules about default error codes and verbosity levels in the doc that (as they say) mostly do the right thing. As an extra special trick, instead of calling die() as our previous Getopt::Long example did when it couldn't parse the arguments successfully, it now calls pod2usage() to spit out the usage message before exiting.

Welcome to My Shell

Just as some people believe that every program that increases in complexity over time eventually grows the ability to send email if it gets complex enough, I think you can make a good case that the more complex command line programs often grow an interactive mode. This interactive mode is usually like a mini-shell. If you find this happens to you, don't panic! Instead, let me offer you a tool to help make your interactive mode more pleasant for the people who will use it.

When building an interactive mode like this, you have to decide what level of help you want from a Perl module. Do you want something to just handle prompt parsing/validation (e.g., using IO::Prompt)? Do you want something to handle terminal interaction so someone can edit her or his commands in place (e.g., using Term::Readline)? Do you want something that will provide a list of valid commands with doc, etc.? Let's see one that gives us the full monty: Term::ShellUI.

Here's the first set of sample code described in the Term::ShellUI doc. I'm showing it to you because it demonstrates a whole host of things about what Term::ShellUI can do and how to do it:

```
    use Term::ShellUI;
    my $term = new Term::ShellUI(
        commands => {
             "cd" => {
                 desc    => "Change to directory DIR",
                 maxargs => 1,
                 args    => sub { shift->complete_onlydirs(@_); },
                 proc    => sub { chdir( $_[0] ||
                                 $ENV{HOME} ||
                                 $ENV{LOGDIR} ); },
         },
         "chdir" => { alias => 'cd' },
         "pwd"   => {
            desc    => "Print the current working directory",
            maxargs => 0,
            proc     => sub { system('pwd'); },
         },
        "quit" => {
            desc    => "Quit this program",
            maxargs => 0,
            method  => sub { shift->exit_requested(1); },
        }
    },
    history_file => '~/.shellui-synopsis-history',
);
print 'Using ' . $term->{term}->ReadLine . "\n";
$term->run();
```

Let's look at the overall structure first. The code creates a new Term::ShellUI object by passing a specification into the module with a few hash keys. Reading from the bottom up to take the simpler one first, you can see we specify history_file, which tells Term::ShellUI to keep a history file. This will make it possible to repeat a previous command (even after you have quit and reentered the program). The more interesting hash key is "commands", the one before history_file. This is where we define which commands our mini-shell will accept and what to do for each command. Let's read from the top down and look at the arguments.

The first command that is defined by this code is a command for changing directories. It has a description to that effect (desc => …), takes a single argument ("maxargs => 1"), provides "tab completion" for its arguments ("args => …", which in this case calls complete_onlydirs() to only offer directory names as part of that completion) and actually performs the command via the Perl function chdir(). The next command, "chdir" shows how easy it is to define another name for a command that will be treated like the original one. The only part of the other commands worth mentioning is the line in the quit command that says:

```
method  => sub { shift->exit_requested(1); }
```

This tells the module to run the exit_requested() method of the object, which sets a flag that requests the module cease asking for more commands. Term::ShellUI has tons of other functionality you'll find described in the doc. Hopefully from this little snippet, it is obvious that you can get a full-fledged interactive mode/shell added to your script with little work.

With that, I hope I've given you a few tools to make more awesome command-line programs. Take care and I'll see you next time.

References

[1] http://www.cryptonomicon.com/beginning.html.

Python: -m Is for Main

DAVID BEAZLEY

David Beazley is an open source developer and author of the *Python Essential Reference* (4th Edition, Addison-Wesley, 2009) and *Python Cookbook* (3rd Edition, O'Reilly Media, 2013). 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

As Python programmers know, Python doesn't really have a notion of a main() function like compiled languages such as C or Java. That is, there's no dedicated function that you define as the entry point to your program. Instead, there is the concept of a "main" program module. The "main" module holds the contents of whatever file you tell Python to execute. For example, if you type this,

```
bash % python spam.py
```

then the contents of spam.py become the main module. For scripts, you might also see the classic #! convention used to make them executable:

```
#!/usr/bin/env python
# spam.py
...
```

Finally, a common idiom found in most code meant to run as a main program is a check that looks like this:

```
# spam.py
...
if __name__ == '__main__':
    # Main program
    ...
```

__name__ is a special variable that always holds the name of the module and is set to '__main__' when executing as a main program. The primary reason for enclosing the main program in such a check is that it allows you to import the file as a library module without triggering main program execution. This can be useful for debugging, writing unit tests, etc.

For many programmers, this is the final word when it comes to writing scripts. I'll admit that for most of the past 15 years, I've never done much more than this or given the idea of a main script much thought. Naturally, there is more than meets the eye, otherwise, I wouldn't be writing about it. Let's dig a bit deeper.

The -m Option

Normally when you run a program, you simply give Python the name of the file that you want to execute; however, a less obvious way to specify the file is as a module name using -m. For example:

```
bash % python -m spam
```

Unlike a simple file name, the useful feature of -m is that it searches for spam on the Python path (sys.path). Although this feature is a minor change, it means that you don't actually have to know where spam.py is located to run it—spam.py merely must be located somewhere where Python can import it.

Once you discover -m, you'll quickly find that there is a wide range of built-in modules and tools that execute in this way. For example, if you want to run a simple Web server on a directory of files, do this:

```
bash % python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

If you want to run a program under the debugger, type this:

```
bash % python -m pdb yourprogram.py
```

Or to profile a program:

```
bash % python -m cProfile yourprogram.py
```

Or to time the execution of simple statements:

```
bash % python -m timeit --setup="import math" "math.cos(2)"
10000000 loops, best of 3: 0.125 usec per loop
bash %
```

Indeed, you'll find that there are a lot of useful things that live behind the -m option. Your application can use it, too. As it turns out, there are several benefits to doing so.

Organizing Large Applications

Almost any non-trivial Python program consists of both library modules and application-level scripts. When you're starting out, putting all of your code in a single directory and not worrying too much about code organization is often fine; however, as things start to grow, you'll want to think about having a better organization than a simple directory with a bunch of files in it. This is especially so if you're going to start giving your code away to others.

For most projects, putting library modules into a package structure is standard practice. You pick a unique top-level name for your project and organize code as a hierarchy. For example, if the name of your project was "diddy," you might make a directory like this:

```
diddy/
    __init__.py
    foo.py
    bar.py
    ...
```

If you've never seen __init__.py before, it's required to mark a directory as being part of a package. The file can be empty, but it must be there for imports to work. Application scripts would then be written to import modules out of this package using statements such as this:

```
# rundiddy.py
# An application script
from diddy import foo
from diddy import bar
...

if __name__ == '__main__':
    # Main program
    ...
```

This approach immediately presents some problems, though. In order for a script like this to work, the related package needs to be properly installed on the Python path (sys.path). This might not be a problem if you're working by yourself, but if you hand the script to a co-worker, it's not going to work unless she also has the associated libraries installed somewhere. As an alternative, you might consider putting the script in a common location (e.g., /usr/local/bin) and telling your co-workers to use that; however, you've now placed yourself in the role of a system administrator as you try to manage the script, the installed libraries, and everything else associated with your application.

All of these problems are caused by the fact that the script and its dependent package are placed in separate locations. As such, you need to worry about path settings, version dependencies, and all sorts of other installation issues. For example, how do you make sure that your script actually uses the right version of its dependent library package? I rarely run into Python coders who haven't ended up creating a big sys.path hacking mess for themselves trying to deal with things like this at one point or another; it can also cause all sorts of weird problems during code development. For example, you're probably not going to get the last few hours of debugging back after you realize that the reason your code is failing is that it was importing a version of a library different from the one you expected.

In-Package Scripts

One nice feature of the -m option is that it allows you to easily create "in-package" scripts. These are scripts that live in the same package hierarchy as the library files on which they rely. For example, you can simply move the rundiddy.py file inside the package like this:

```
diddy/
    __init__.py
    foo.py
    bar.py
    rundiddy.py
    ...
```

Once a script lives in a package, you can additionally modify it to use package-relative imports like this:

```
# rundiddy.py
# An application script
from . import foo
from . import bar
...
```

If you've never seen a package-relative import before, the syntax from . import foo means load foo from the same directory. Similarly, a statement such as from .. import foo loads a module from the parent directory whereas from ..utils import foo loads a module from the directory ../util relative to the module doing the import. I must stress that this syntax only works within a proper package—you can't use it in arbitrary Python modules. Additionally, you're not allowed to write an import that "escapes" the top-level package directory.

One nice thing about package-relative imports is that you no longer need to hard-code the top-level package name into the source, meaning that renaming the top-level package to something else is easy. For example, if you need to have two different versions of your package installed at the same time, rename one of them (e.g., "olddiddy"). All of the imports within the package will still work if they've been written using the package-relative style.

To run an in-package script, you simply type python -m diddy.rundiddy. If you've done things correctly, the script will simply find all of its correct library files, with no path hacking or installation headaches.

If you're put off by having to type python -m diddy.rundiddy, you can change the name of the rundiddy.py file to __main__.py. You'll then be able to type python -m diddy and it will simply run the __main__.py file for you. (As an aside, few programmers realize that any directory of code with a __main__.py file can be directly executed by Python.)

Who Cares?

The main benefit of moving scripts inside a package is that they effectively allow you to create a kind of code bundle where everything is self-contained. For example, if you wanted to give your application to a co-worker, you could simply hand them the top-level directory along with instructions on how to run the code (using -m). If you've done everything right, the code will simply "work" without ever having to fiddle with path settings, installing code into the user's Python installation, or anything else. During software development, this is actually a really useful thing—you can hand someone your code and have him try it out without requiring him to muck around with his local Python setup. Similarly, if you're working on a new version of code, you can do it in your own directory without ever worrying about previously installed versions getting in the way. Again, the key thing that makes this possible is the fact that everything is bundled together in one place.

I've found this approach to be useful in writing various application-level tools. For example, consider this hypothetical application structure:

```
diddy/
    __init__.py
    foo.py
    bar.py
    __main__.py
    server/
        __init__.py
        httpserver.py
        rpc.py
        message.py
        __main__.py
    worker/
        __init__.py
        queues.py
        request.py
        __main__.py
```

Within this directory, there are actually three separate "applications" that are executed using -m. For example:

```
bash % python -m diddy          # Executes diddy/__main__.py

bash % python -m diddy.server    # Executes diddy/server/__
main__.py

bash % python -m diddy.worker    # Executes diddy/worker/__
main__.py
```

Again, it's a self-contained bundle of code. There are no scripts to install and no path hacking to be had other than making sure the top level "diddy" directory is available when you run Python (it could be in the current working directory).

Application to Testing

Another place where I've found the package approach to be useful is in unit testing. A problem I always seem to face is figuring out how to make my unit tests use the correct version of code. That might sound silly, but I can't count the number of times I've run some tests only to find out that they executed using a completely different version of the code than the one I was working on due to some kind of sys.path issue. In response to such problems, you might be inclined to hack sys.path in some manner. For example, in one of my projects, if you look at the testing files, the first thing the tests do is hack sys.path to make sure the tests run using the right code base. Frankly, it's clumsy and a bit embarrassing.

As an alternative, you can move the tests inside the package and use the -m option to run them. For example, consider a project with this file structure:

```
diddy/
    __init__.py
    foo.py
    bar.py
    tests/
        __init__.py
        foo.py
        bar.py
        __main__.py
```

In this organization, the tests directory mirrors the structure of the package itself. Each testing file is a stand-alone executable that looks like this:

```
# tests/foo.py
import unittest
from .. import foo

class TestSomething(unittest.TestCase):
    def test_example(self):
        result = foo.do_something()
        self.assertEqual(result, expected_result)

    # More tests follow
    ...

if __name__ == '__main__':
    unittest.main()
```

To run a single testing file, you simply type a command like this:

```
bash % python -m diddy.tests.foo
.....
----------------------------------------------------------------------
Ran 5 tests in 0.295s

OK
bash %
```

I might reserve the tests/__main__.py for running all of the tests at once. For example, a simple approach is as follows:

```
# tests/__main__.py
from .foo import *
from .bar import *

if __name__ == '__main__':
    unittest.main()
```

Now, tests can be run like this:

```
bash % python -m diddy.tests
.........
----------------------------------------------------------------------
Ran 9 tests in 0.423s

OK
bash %
```

Saying whether such approach would appeal to hard-core testing experts is difficult; some might argue that the tests should be contained in their own dedicated directory separate from the package itself. To be sure, this might not scale for a tremendously huge project. Nevertheless, I've often found this approach to be simple, reliable, and quite effective in medium-scale projects. Part of the appeal is that it works without having to fiddle around with the environment or a complex set of extra tools. Of course, your mileage might vary.

Closing Words

Every so often a feature of Python comes along that really catches my fancy. The -m option definitely falls into that category as I find myself using it more and more. Honestly, the main appeal of it is how it allows my scripts and library code to be bundled together into a single cohesive package. As such, it saves me a lot of time where I would have to be fiddling around with path settings and installation issues. No, life is too short for that. Instead, put everything in a package and use -m. You'll thank yourself later.

iVoyeur
Go, in Real Life

DAVE JOSEPHSEN

Dave Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.
dave-usenix@skeptech.org

Through a combination of unfortunate timing, unexpected workload, and laziness, I'm writing this column in the midst of a rare vacation, as I look out on the eastern front of the Rocky Mountains in late fall. I'm using a borrowed laptop (thanks Chris) in a land unencumbered by WiFi, and I'm hoping to find a GPRS signal strong enough to send it to Rik, my editor, before the deadline, which is today I think, or maybe tomorrow.

Although we've arrived only a few weeks later in the year than usual, everything is different here in my favorite place in the world: the air colder, the animals edgier, the light and foliage more dramatic. When we manage to make it up here, we expect to be snowed on at least once or twice, but this time we've been either rained, iced, or snowed on every day. This has only accentuated our hiking, affording us some privacy on the trails, increasing the contrast of our photos, and giving our supposedly waterproof boots an opportunity to prove their worth.

I love the mountains, not just because their size puts humanity in perspective, and not just because they are unabashedly wild. I love the mountains because they encourage good habits in the people who choose to venture into them. They reward hard work, awareness, and respect, and they punish stupidity, sloth, and arrogance. I love the mountains because loving them makes me a better human being.

I had planned this month to write more about libhearsay [1], and show off how I've used it to connect a few different monitoring tools together. But that work is 3000 or so miles away, and anyway those ideas could stand to be baked a bit more before I force them upon you like an excited co-worker with a USB-stick full of vacation slides.

Instead, because I've been writing libhearsay in the Go programming language and also because Go is a newish and hotish programming language created from scratch by the likes of Ken Thompson and Rob Pike, I thought I'd share my experience with it thus far.

In the past few years, many smart programmers have written a bunch of brilliant articles about Go that cover every nook and cranny of every feature and function. None of them, however, seem to convey a sense of what it feels like to create a program in Go, especially from the perspective of a systems guy rather than an application developer. Having worked with it for a few months and a couple of thousand lines, I've noticed that, like the mountains, Go seems to be encouraging beneficial habits in me. Some of these are small things, and are easily articulated, and others are larger and more subtle, but taken together, the patterns, idioms, and manners of thought that Go encourages are making me a better programmer. I think that this, rather than any particular linguistic feature, is the second greatest thing about Go (the greatest thing about it obviously is its enormous potential for name-related puns). Here are a few examples:

Go encourages me to use Git.. ahem, from the git-go

The "go" utility, which is a combination compiler, linker, and packing tool, expects my Go code to be organized into a simple directory structure. If I place a github.com folder inside the top-level src directory of this structure, and commit the contents of a subdirectory of the GitHub folder to GitHub, then other Go users can install and build my program by typing "go get foo" at their command prompt (where "foo" is the name of my project on GitHub).

The go utility will go to GitHub, find my project, clone it into the local users $GOPATH/src/github.com folder, and build it for them. This is pretty great; you get a handy packaging mechanism for free by using revision control, which is something you would have done anyway. It supports sites other than GitHub, such as launchpad, googlecode, and bitbucket, and a slew of version control systems, including Git, Mercurial, Subversion, and Bazaar. You can even use private sites by following a naming convention or by providing a <meta> tag.

The scheme is not without its problems, including, perhaps ironically, that it's not easy to specify upstream package versions, but it's also illustrative of the underlying pragmatism that typifies Go as a language. The developers didn't bother coming up with an unwieldy reimplementation of CPAN or Gems; instead they observed that developers like to keep code in revision control systems and hacked up a simple, lightweight package manager as the shortest path to getting developers what they probably want anyway.

Go encourages me to think about concurrency

Despite the hours (days?) of study I've invested in my considerable understanding of threading models and inter-process/inter-thread communication libraries, and despite the tens (hundreds?) of little test programs I've written in C, Perl, Python, and Ruby in my attempt to implement those models, and even despite the multi-threaded/multi-process open source projects to which I've committed code, I have never once in my professional life written a concurrent program for use in production. Nor have I ever revisited and rewritten one of the thousands of little tools I've written to make it concurrent. Not that is, until I met Go.

This is not for lack of understanding or caring on my part. In real life I'm an OPS, and the nature of the job just makes impractical the creation of multi-threaded tools to solve the mundane sort of everyday problems that I run into (at least in the shops I've worked in so far). There is neither the time nor the payoff. This sucks for me, because it means I don't get to think concurrently often, and as I grow older, it probably renders that sort of thinking more difficult for me. So that's awesome; my current languages are destroying my brain.

The second Go program I ever wrote was concurrent. It was not concurrent because I wanted to prove or understand the model, or because I was bound and determined to use go-routines and channels. It became concurrent naturally, as a result of my problem and the fact that go-routines were available. Go-routines are so handy that functionally, their use is hard to avoid. Which brings me to:

Go encourages me to network

In the past, for example, I would avoid putting socket code into my tools. I've written socket programs for my own edification, and fully understand the threading issues among others, but in real life it almost always makes more sense to quickly hack up something to standard I/O and rely on daemontools, for example, for TCP. This sentiment is alive and well among the node.js crowd these days, but it is simply no longer true with Go. The concurrency features are so well implemented that there is no reason not to roll your own TCP server.

For anything of moderate size that is expected to remain resident in memory, there's no reason not to roll your own HTTP server for that matter, and it's pretty common practice among Go developers to build something like a distributed worker daemon in Go, and then add an HTTP server to it to export metrics and state data, or add an interface to control the worker remotely.

Go encourages me to embrace type and think about data structures

In Go, creating your own type and extending it with a method is so simple that even as someone who has never been enamored of OOP, or the concept of sub-classing, I find myself naturally reasoning about my solutions primarily in terms of the interaction between custom types. I think Go makes this palatable to me because there isn't any ceremony or magic involved. Type creation is no different from typedeffing in C, and adding methods to types is only trivially different from function declaration.

As a result, where in any other language I might create an array of doohickeys, and loop across them doing whatever, like:

```
for(i=0, i<numberOfDoohickeys,i++) myDoohickey=listOfThings[i]
doWhatever(myDoohickey)
```

in Go I'm much more likely to create a doohickey type of my own to store in the array (which is probably a pretty complex (for me) nested type), which has a built-in whatever method like this:

```
for i in listOfDoohickeys i.Whatever
```

I know, those pretty much seem like the same thing, but by creating my own doohickey I get to think about lots of interesting things, such as exactly how large a doohickey is in memory and whether the system creates a copy of my doohickey in memory when it performs the whatever function, or operates directly on the existing doohickey via a pointer.

It also means that, although a program that loops across some doohickeys doing whatever is useful maybe once or twice, a program that defines doohockeys and implements an interface to them that does whatever is useful may be a lot longer, because other developers (or I) can come back later and trivially add more interfaces to do other things. Now we have a shop-wide means of dealing with doohickeys, and everybody who does whatever to a doohickey from now on will do it in a repeatable way without having to reinvent the wheel.

There's an xkcd comic [2] where, having been asked to pass the salt, an off-frame OCDish person begins developing a general interface that will enable him to pass arbitrary condiments, and over-engineering like this can easily get out of hand in some of the other languages I've used. But I've noticed that general interfaces spring into being quite naturally in Go without any grand intention or purpose on my part; I didn't whiteboard an interface for doohickeys, or prototype it in a simple language and then properly reimplement it in another. I didn't begin by creating a doohickey library or subclassing something doohickey-like. I—a meathead, knuckle-dragging OPS—in scratching my own immediate doohickey itches, tend to accidently create robust, probably even concurrent engineering solutions in Go. Solutions that other OPS are likely to thank me for. As someone who has, for years, prefaced my scripts with something like:

```
#Blame Dave: Fri Sep 15 20:56:47 CDT 2006
```

I appreciate creating code that I don't need to feel vaguely guilty about.

Finally, in other languages I've used, a certain amount of risk came along with simplifying things like sockets; a linear relationship between the language's ability to expose cool features and the amount of cruft in my own code as I bolted on this or that. I had to keep things simple, so the program execution remained knowable—and this is perhaps unfortunate, because what is the point of having a simple interface to sockets if you always feel like it's too cognitively expensive or ugly and bloated to use?

In Go, however, the type system has a tendency to keep everything clean and compartmentalized. My Go code is resistant to cruft. If you aren't fighting it, the code naturally segments and documents itself via its type and function definitions, so adding something like a TCP server doesn't clutter things up, and more importantly, doesn't make your types—and therefore your program—any more difficult to reason about. To be clear, I'm not throwing HTTP servers into everything I write just in case, but I'm certainly more likely to add something like a network interface to expose some analytics where it makes sense to do so.

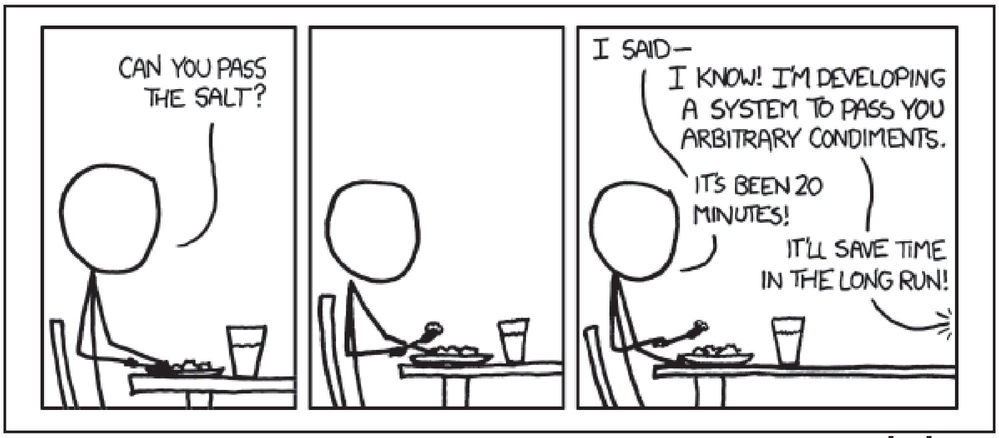
I'm painfully aware that most of what I've said in this article amounts to subjective drivel that could probably be repeated en masse by any proponent of any programming language ever, so even though it won't help, I'll mention that I'm not married to Go and, in fact, program in a multitude of languages. My intent here was not to steal anyone's mindshare or compliment Go at the expense of any other language in particular. But I will wholeheartedly suggest that you learn Go if you get a chance. If you start using it, I think you'll notice that Go wants you to be productive. It keeps things simple, stays out of your way, rewards you for being you, empowers you to build interesting stuff, and makes you a better programmer in the process.

References

[1] libhearsay http://www.skeptech.org/hearsay.

[2] http://xkcd.com/974/.

xkcd


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

xkcd.com

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

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 te chnical sessions. This is perfect for folks on the go or those without consistent Internet access.

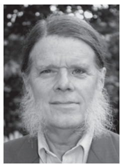
Box Sets are available for:

- **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 Mangement 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

Measuring vs. Modeling

DAN GEER AND MICHAEL ROYTMAN



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

Michael Roytman is responsible for building out Risk I/O's predictive analytics functionality. He formerly worked in fraud detection in the finance industry, and holds an MS in operations research from Georgia Tech. In his spare time he tinkers with everything from bikes to speakers to cars, and works on his pet project: outfitting food trucks with GPS.
mikeroytman@gmail.com

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

—Sir Arthur Conan Doyle, 1887

Punchline: Using CVSS to steer remediation is nuts, ineffective, deeply diseconomic, and knee jerk; given the availability of data it is also passé, which we will now demonstrate.

Vulnerability data is often used to describe the vulnerabilities themselves. This is not actually interesting—it's like using footprints to describe bear paws. Sure, a black bear has different ones from a polar bear . . . but a more interesting fact is what kind of fur they have.

Strategies for vulnerability remediation often rely on true, but irrelevant, facts. The problem begins with how vulnerabilities are defined. There are several places that define vulnerabilities, but Common Vulnerabilities and Exposures (CVE), while not the most complete, is the most universal set of definitions with which we have to work. Yet thinking of CVEs as elements on the periodic table is a grave mistake; before creating synthetic polymers (read: useful analytics) out of these elements, we need to understand the biases and sources of uncertainty in the definitions themselves. For example, take a look at this finding from a research team at Concordia University in their 2011 paper "Trend Analysis of the CVE for Software Vulnerability Management" [1]:

"Our finding shows that the frequency of all vulnerabilities decreased by 28% from 2007 to 2010; also, the percentage of high severity incidents decreased for that period. Over 80% of the total vulnerabilities were exploitable by network access without authentication."

There are many such papers out there and they may be guiding organizational decision-making, but, to our point, that type of analysis misses the boat on what is being analyzed. An increase or decrease in vulnerability frequency or the enumeration of vulnerability types seen in successive time intervals can have wildly varying biases. CVE is a dictionary of known infosec vulnerabilities and exposures. It is a baseline index for assessing the coverage of tools; it is not a baseline index for the state of infosec itself.

Looking at the volume of CVEs seems to suggest that steadily increasing CVE disclosures mean "the state of security is getting worse" or some similar inference. However, CVE is not a dictionary. It is from a company attempting to streamline a process with limited resources. If you want to understand why the unit of risk we're so used to isn't a unit at all, take a look at Christey and Martin's "Buying Into the Bias: Why Vulnerability Statistics Suck" [2]

CVSS, the most widespread vulnerability scoring system, is a model for scoring the relative likelihood and impact of a given vulnerability being exploited. Among other inputs, the model takes into account impact, complexity, and likelihood of exploitation. Next, it constructs a formula based on these by fitting the model parameters to a desired distribution. This comment was made during the drafting of CVSS v2:

"Following up my previous email, I have tweaked my equation to try to achieve better separation between adjacent scores and to have CCC have a perfect (storm) 10 score…There is probably a way to optimize the problem numerically, but doing trial and error gives one

Week	CVEs affected	Breach count
1	67	754588
2	13	191
3	4	157
4	18	3948
5	15	9361
6	81	62307
7	70	41619
8	71	39914

Table 1: Breach traffic June–August 2013

plausible set of parameters...except that the scores of 9.21 and 9.54 are still too close together. I can adjust x.3 and x.7 to get a better separation . . ." [3]

So what facts is this model twisting? Well, for one, at the time of the creation of the model, there was no data available about the likelihood of an exploit. Today, we have SIEM logs with CVE attack pattern signatures, and most enterprises have both a vulnerability scanner and a SIEM installed on their networks. This allows us to correlate a CVE to the attack signature and track exploits. No need to blame the model, it's just that the theory was created, as Sherlock so aptly put, before there was any data. Moreover, when a CVE gets a score, an analyst does some research, and assigns a point-in-time likelihood value.

We can do better than that. The biggest problem with the CVSS model is not the way in which it is executed but rather what it seeks to expose. It is trying to capture (in the temporal component) a snapshot of what the live instances of attacks against these vulnerabilities look like—but it is attempting to do so without looking at any live data. Instead, the CVSS model is a static definition of the very stochastic process of exploit and breach traffic.

The present authors have access to 30 million live vulnerabilities across 1.1 million assets (hostnames, IPs, files, URLs) and 10,000 organizations. Additionally, using a different data set of 20,000 organizations' SIEM logs, analyzing them for exploit signatures, and pairing those with vulnerability scans of the same environments (data collected on the Open Threat Exchange), we construct a stochastic picture of breach traffic over the months of June to August 2013, affecting the 135 unique CVE identifiers that presented themselves in that period. No possible interpretation of that data (see Table 1) lends itself to a static conception of likelihood of exploit.

This is where the correlation gets fuzzy. The breaches come from a different set of organizations than the live vulnerabilities we have access to. However, as the sizes of both sets get bigger,

CVSS score	CVSS v1 Pr(breach)	CVSS v2 Pr(breach)
1	0.210%	0.210%
2	-0-	0.36%
3	-0-	-0-
4	1.033%	0.480%
5	0.642%	1.220%
6	0.266%	0.220%
7	0.102%	0.070%
8	0.811%	1.432%
9	2.283%	2.438%
10	4.726%	3.530%

Table 2: Probability of exploit using CVSS as the measure

the conclusions we can draw from the correlations between them gain significance. Because this is observed data, per se, we contend that it is a better indicator than the qualitative analysis done during CVSS scoring.

How much better? Let's assess a couple of possible strategies for choosing which vulnerabilities to remediate. If one chooses a vulnerability at random from the set of possible vulnerabilities, then the probability that a breach has been observed via that vulnerability is roughly 2%. This is our baseline. In Table 2 we show the probability of breach for vulnerabilities with particular CVSS scores, which pale by comparison to the probabilities of breach for vulnerabilities with entries in Exploit-DB or Metasploit or both as seen in Figure 1.

Luca Allodi from the University of Trento [4] has already done this type of analysis on the definitional level. Correlating the National Vulnerability Database (NVD) to the Symantec Threat

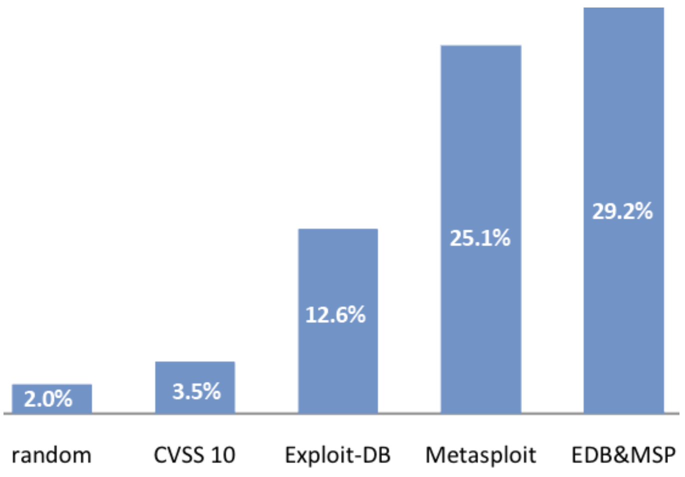


Figure 1: Probability of exploit using other measures

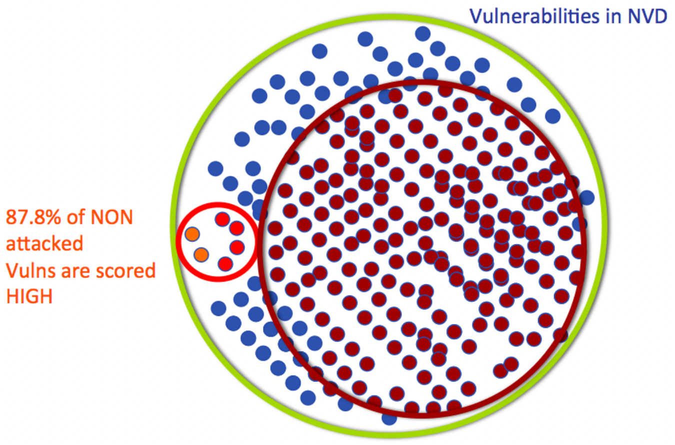


Figure 2: Attacks vs. CVSS score

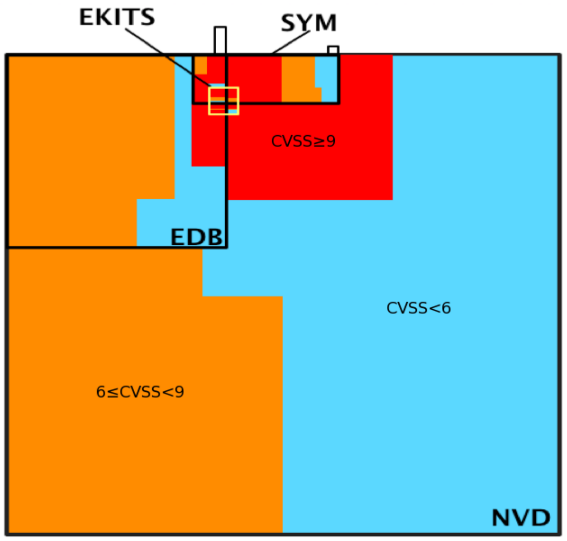


Figure 3: CVSS scores vs. EDB/EKITS/SYM/NVD

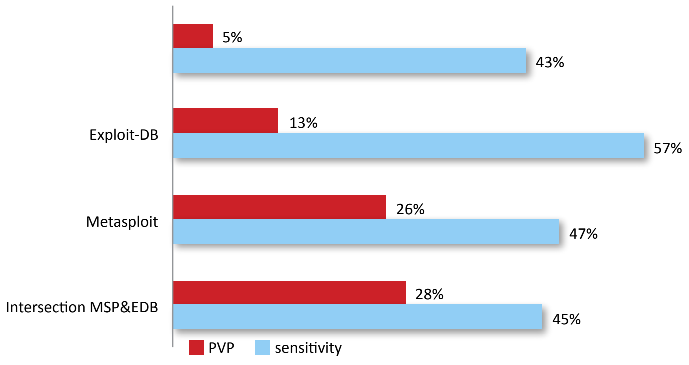


Figure 4: PVP & sensitivity comparison

Exchange in Figure 2, the outer circle encloses all NVD vulnerabilities, the smallest circle is the 2.4% of the NVD vulnerabilities that are actually attacked, and the larger interior circle represents the 87.8% of vulns that are scored ≥9 but are not attacked—which is the point: a high CVSS score does not imply impending risk in need of immediate mitigation.

Allodi's research further correlates the data with Exploit-DB and EKITS (an enumeration of CVE entries in blackhat exploit kits). Figure 3 reproduces his diagram of CVSS scores stacked against Exploit-DB, EKITS, and Symantec's Threat Exchange (to be meaningful, this figure must be viewed online at: https://www.usenix.org/publications/login/december-2013-volume-38-number-6). Dimensions are proportional to data size; vulnerabilities with CVSS ≥9 are red, vulnerabilities with 6≤CVSS<9 are orange, and vulnerabilities with CVSS<6 are cyan. The two rectangles outside of NVD space are vulnerabilities not present in NVD.

There are many entries with CVSS≥9 but with no exploit nor even any live exploit traffic. Conversely, a large portion of Exploit-DB and Symantec's intelligence go unflagged by CVSS scoring; however, this is still a definitional analysis. Visually, it is easy to see that currently adopted strategies—namely, the pervasive use of CVSS to direct remediation [5]—yield undesirable false negative rates (false positives rates are commonplace and widely accepted in remediation strategy). What is of greater interest, however, are the false positive and false negative rates of remediation strategies based on live vulnerability analysis.

Two terms of art from diagnostic testing are *predictive value positive* (PVP), the proportion of positive test results that are true positives, and *sensitivity*, the proportion of true positives that test positive. Using the same data set as above, in Figure 4 we can now really see the value of measuring vs. modeling.

Not everyone has the kind of large scale data we have here, so what is a CISO to do? First, remember that a model is a model—understand the implications of that by collecting some data on yourself, and make a commitment to long-term longitudinal data collection. Assess how well your remediation strategy is performing against your adversaries—adversaries do this all the time; they will implement different exploit kits or simply target others if the success rates of their kits decrease. Some black-market exploit kits offer SLAs to their customers with refunds if the attacks are detected or unsuccessful. A good way to do your assessment is to use an incident response team as a way to obtain the kind of predictive value positive metrics you see above. Use more than one indicator for whether to spend the labor to remediate a particular vulnerability (as we also illustrated above). For the C-suite, being able to show a metric about the level of effectiveness of a program is important, but more important is being able to claim a reduction in the volume of data that the security team has to sift through to get to similar results. In our data set, while the intersection of ExploitDB and Metasploit yields a marginally better sensitivity, the predictive value positive is far higher, indicating that to get the same results, the "cost" is reduced. This is a metric that is useful in practice and accessible to the C-level.

This column suggests a few measures for an efficient, impactful security practice. It is probable that there are other attributes of a vulnerability which are better indicators of breach or which increase operational efficiency. The 28% PVP we obtain here is relatively inefficient even if much better than prior art. Identifying these attributes and using them to generate better predictive metrics is key to more effective security practices.

References

[1] "Trend Analysis of the CVE for Software Vulnerability Management": dc239.4shared.com/doc/JAFW1G95/preview.html (tinyurl.com/k57gxqe).

[2] Steve Christey and Brian Martin, "Buying Into the Bias: Why Vulnerability Statistics Suck":

www.attrition.org/security/conferences/2013-07-BlackHat-Vuln_Stats-draft_22-Published.pptx (tinyurl.com/ksalk3z).

[3] Appendix D: CVSS 2.0 Base Score Equation: www.first.org/cvss/history#c8 (tinyurl.com/mex8a2x).

[4] Luca Allodi, "Risk Metrics for Vulnerabilities Exploited in the Wild": securitylab.disi.unitn.it/lib/exe/fetch.php?media=seminar-unimi-apr-13.pdf (tinyurl.com/p252aa2).

[5] Guide to Adopting and Using the Security Content Automation Protocol (SCAP) Version 1.0: csrc.nist.gov/publications/nistpubs/800-117/sp800-117.pdf (tinyurl.com/m3famtj).

/dev/random
Cloud Control: Future (Mis)Directions for Information Security

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

A columnist of my ilk (A positive, with vitamin D and sodium benzoate added to prevent spoilage) realistically has only three basic choices of topic: what's going on with some aspect of technology now, what went on with some aspect of technology in the past, or what might happen with some aspect of technology in the future. I've churned out a fair amount of slush on the first two, so now it is time to offer my insights, such as they are, on what someday might have been.

Security is a crap shoot in the best of times, or maybe a roulette wheel. A roulette wheel fixed strongly in favor of the house; the house, as usual, being controlled by various unsavory elements (Sleazium, Larcenium, Felonium, et al.). Security is an abstract concept, unwieldy and unworkable in the real world. The bottom line is that where the rubber meets the road in the final analysis at the end of the day, "security" is overused and under-defined.

What we really mean when we talk about security is in fact "insecurity." A secure system is one that has not yet been designed and built; all secure systems are therefore future systems. Systems currently in operation, ipso facto, are inherently insecure, or at best both secure and insecure simultaneously. Taking the quantum superposition comparison further, any attempt to characterize the security of a system causes that duality to break down. Heisenberg would appreciate that you can never really calculate how secure your system is, only the probability that it has been compromised today. Or, for the purposes of this discussion, tomorrow. Security is Schrödinger's cat, long-deceased and skeletal.

Now that I've cleared some of the more egregiously tattered metaphors and dog-chewed aphorisms out of my virtual writing desk, I can relax and try to make some sense. The term "security" has, to paraphrase James Thurber, taken a terrific tossing-around in recent years and no longer means much of anything. There is no "security" in "information security;" there is only risk and the mitigation thereof. Risk management is where my professional attention is now directed because security is not something I know how to achieve. Those of you who are privy to the duties of my "day" job will understand. The rest can talk quietly amongst yourselves until the bell rings.

This trend toward increasingly draconian measures to self-identify to your software and hardware has just about reached its practical limitations, from what I can see. As Apple recently had the shattered pieces of its much-touted thumbprint authentication process for the iPhone 5S handed back to it in a paper bag by Germany's Chaos Computer Club, so will likely go most major "innovations" in access control for the foreseeable future. There is nothing you can possess, Dr. Jones, that I cannot emulate.

In my version of the future, authentication will move from the I/O device to the cloud. To authenticate, you tell the Master Interrogator Interface who you claim to be and three people you claim not to be. Once it verifies all four claims, you are granted access. That sounds perfectly potty, of course, but is it really any sillier than most other authentication protocols? I think not.

Maybe we'll see reliable whole-body photorecognition come into its own, as well. Those "selfies" you like to snap may someday get you access to your money or home entertainment/security system. Perhaps there will be a software photo mask that keys on a unique micro-attribute like your pore structure or acne scars.

Application security today is haphazard and depends mostly on programmers not making any of a dozen or so major blunders in their code: no bounds checking, relative paths, formatting errors, and so on. In the future, I predict that applications (which now reside solely in the cloud) will have no security measures at all taken during their coding. Because applications themselves will be modular with extreme granularity and distributed across the cloud, each instantiation of a particular program will be unique. Anti-malware functions will be provided by heuristically programmed agents in the cloud that watch for and forbid anomalous and/or dangerous behaviors. Antivirus companies will no longer sell subscriptions to signature files. Instead, they will activate their cloud heuristic agents for a set period of time for a specific customer . . . for the traditional hefty fee.

Thunderheads will be the airborne pathogens infesting the future cloud, much as Blackhats are the venomous spiders in today's Web. MITM will stand for "Man in the Miasma," because "middle" isn't very descriptive or accurate in a structure as amorphous as the cloud. Hackers will cease to have "handles" but will instead adopt "tail numbers." Being positively identified will be to "Fall Out" (of the cloud). Wags will call this "precipitation," but wags will always be wags.

Encryption, rather than referring to data scrambled by a complex algorithm that requires a lengthy key to reconstitute, will denote information that is actually en-crypted. That is, it will be encased in a cocoon built of layers of nonsense information that can only be penetrated and the data transcribed using the mathematical equivalent of a biological polymerase. Not only must the correct transcriptase be used, but the start and stop codons must also be correct—as must even the rate of transcription—or the message will not be comprehensible. Presuming, of course, it was comprehensible to begin with.

Web defacement, that exceptionally juvenile scourge of the late '90s and early '00s, will be replaced by attacks known as ODEs: On-the-fly Drive-by Exploits. As application code modules are assembled to order in the cloud, fragments of exploit code—able to hide from the heuristics agents by dint of being non-functional on their own—self-assemble into unique malware, the ultimate functionality and virulence of which depends on the identity and assembly order of the constituent fragments.

In closing, I'd like to stumble over Advanced Persistent Threats. I say "stumble over" because my favorite example of an APT is my cat. She's advanced—easily as smart as a toddler and much more creative; persistent—if she wants something, she is simply not to be ignored; and a threat—she's fond of sprawling across the narrow pathway I take every morning before dawn to get to my shower. But never in the same place twice. The fact that I have managed to remain fracture-free, proud to be, for 13 years as her roommate and food provider is nothing short of a miracle.

Thaumaturgy, coincidentally, is my candidate for the most robust information security tool available.

Book Reviews

ELIZABETH ZWICKY AND MARK LAMOURINE

Data Science for Business
Foster Provost and Tom Fawcett
O'Reilly Media, 2013. 366 pp.
ISBN 978-1-449-36132-7
Reviewed by Elizabeth Zwicky

Data Science for Business is an introduction to data science as it is applied to business. The book includes enough information to tell you what you can do, how you can do it, and whether or not your data scientists are crazy and/or making things up to impress you. If you have exposure to machine learning already, this book is enough to give you a start on how you can expect the real world issues to go. You'll need another book if you want to actually implement these ideas, but those books are easy to come by.

The authors clearly have real-world experience, both with the kinds of misery that occur in real data sets and with the common problems of academic data analysts encountering them. Many beautiful theories do badly when faced by the messiness of real data and questions, and worse yet, they do badly in subtle ways. Producing apparently good results that are in fact pointless or actively bad is easy, which makes for a lot of tension between data scientists and the business groups they are working with.

This is highly technical stuff, and explaining it to people who are not mathematicians or computer scientists without oversimplifying it is hard. The authors do a nice job of simplifying it just far enough. You still must be willing to think about abstract concepts with numbers in them, and not to be intimidated by mathematical symbols, but understanding the topic doesn't require higher math. (On the flip side, if you know all the math already, you're going to need a tolerance for simplified notation.)

I'd recommend this book to people on either side of the business group/data scientist relationship, or to people trying to understand what data science can realistically do for their organization. Somebody who understood this book would meet my group's criteria for "not a loser" about data science or machine learning.

You'd need a bit more to get to "useful in the fraud space," including an understanding of why you could have an overfitted model and still need more features. In case you're curious, overfit is caused by the number of features you use, not the number you have, and is worsened by having poor quality features, plus it is heavily dependent on sample size. On big data, overfit is often not reached until >100 features, which the authors do point out, but they fail to mention that you may need twice that many to get enough good ones. Naive data scientists with first-order understanding are almost always too afraid of overfit and not focused enough on features. Still, somebody worried about overfit is easier to deal with than somebody who happily reports that they have an excellent model, because it performs perfectly on the data it was built on. People really do this, to my amazement. They also really build models based on the feature they're trying to predict. This book explains how not to.

Hiring the Best Knowledge Workers, Techies & Nerds
Johanna Rothman
Dorset House, 2004. 330 pp.
ISBN 978-0-932633-59-0
Reviewed by Elizabeth Zwicky

Hiring the Best Knowledge Workers, Techies & Nerds lays out a humane and effective hiring process, including a way of developing job descriptions that actually results in something practical for all players. This is a recent digital release, which is how it caught my eye. The publication date is mostly irrelevant, although occasionally it is noticeable. Do job seekers still consult newspapers? They must somewhere, but not in Silicon Valley.

This title would be most useful for somebody in a small company because it is geared toward people with relatively little assistance in the hiring process. But the sections on writing job descriptions, tailoring them for specific places they may be used, evaluating resumes, and training junior people to interview are useful to most people, regardless of their environment. The author's suggestions are nicely balanced, and she provides good stories to illustrate why you want to treat job seekers with generosity, not rejecting people blindly for typos and mistakes. She does not mention the contentious topic of thank you notes at all.

I found it a little slow-starting, and worried at the beginning that it would all be too HR-speak, but I warmed up to it. I would have liked more discussion of cases in which you have a ton of hiring to do and may not be interested in tying yourself tightly to a specific job description, but I think the book will still be useful to most people who need to hire technical people.

Why We Fail: Learning from Experience Design Failures

Victor Lombardi
Rosenfeld Media, 2013. 214 pp.
ISBN 978-1-933820-17-0

Reviewed by Elizabeth Zwicky

Other people's disasters are always amusing, and *Why We Fail* presents a fine collection of products that failed even with plenty of starting advantages. The author ties them together as failures to delight the customer with the experience, and traces down the reasons why the experience came out lacking. He believes that better testing of the experience and the assumptions being made will fix the problem; I'm not so sure, as several of these companies seem to have made explicit decisions to favor other factors. Possibly more testing would have helped designers better understand the cost they were paying, but it's also possible that they would have made the same bets anyway.

Still, whether you agree with the author's conclusions or not, there's a lot to think about in this book. If nothing else, you should come away convinced that technical excellence, being first to market, being well funded, and being the existing market leader are not enough to save you from disaster, whereas engaging with and delighting the customer might. You will probably also be convinced that if you want to delight the customer, you should find one, give her the experience, and believe what she tells you about it.

And, if your next project fails, you'll at least be able to console yourself with the idea that you are in excellent company.

Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior

Tom DeMarco, Peter Hruschka, Tim Lister, Steve McMenamin, James Robertson, and Suzanne Robertson
Dorset House, 2008. 234 pp.
ISBN 978-0-932633-67-5

Reviewed by Elizabeth Zwicky

Adrenaline Junkies and Template Zombies is an excellent book for somebody who is starting to realize that project management involves a lot of interesting stuff that is not covered in normal project management books. Somebody beginning to ask questions such as "Is it me, or is this somehow a train wreck in progress?" and "Why do some teams just work better than others? Can it actually involve chocolate and foam weaponry?" and "Can a really great team still be the problem?" (Probably it's a train wreck, yes the frivolity is causally related to the excellence, and yes, teams can be misplaced.)

For me, this book is amusing, but not transformative. I recognize many of the patterns and anti-patterns, I appreciate the authors' willingness to acknowledge that issues of fit mean that something can be a positive or a negative depending on its surroundings, and I got the satisfying feeling of having experiences fall into patterns and seeing them with new eyes.

Tableau Data Visualization Cookbook

Ashutosh Nandeshwar
Packt Publishing, 2013. 152 pp.
ISBN 978-1-84968-978-6

Reviewed by Elizabeth Zwicky

Tableau Data Visualization Cookbook is not a cookbook; rather, this book is a manual arranged by concept. If it were about food, this book would have "recipes" with titles like "Dicing" and "Sautéing." (To be fair, your average cookbook in computing would have recipes like "Egg-based sauce" with a brief example about Hollandaise and a passing mention of Eggs Benedict as an example use, with a cross reference to a recipe for toasting things and possibly one for poaching eggs. Nobody wants the computer equivalent of a cookbook, really.)

A cookbook approach would make sense if the manuals were terrible, but they aren't. They are more oriented to Tableau's view of the world (you have to look up graphing things under "Building Views"), but they are better illustrated, and I prefer their format. The cookbook layout involves bars that unfortunately highlight the unchanging section headers "Getting Ready" and "How to do it . . ."

The book is not terrible, and if the manuals don't meet your needs it may help you out. *Tableau Data Visualization Cookbook* has some interesting tricks, but on the whole I was disappointed. In a cookbook format I expect either a lot of information about specific issues (like the *Regular Expression Cookbook*) or an approach that illuminates the peculiarities of the software by using tasks (like the *R Cookbook*). *Tableau Data Visualization Cookbook* is closer to the latter, but doesn't carry its examples through with enough detail or independence to get there.

Vagrant: Up and Running

Mitchell Hashimoto
O'Reilly Media, 2013. 156 pp.
ISBN 10:1-4493-3583-7

Reviewed by Mark Lamourine

Desktop virtualization has been around for about a decade now. Until recently the desktop VM systems have focused on how to create a VM and then how to power it on and off. Little details like OS installation and network configuration were generally left to the system administrator to manage just as they would have been for physical hosts. Vagrant lets users ignore or automate those tasks as they wish.

In eight brief chapters, Hashimoto fulfills the promise of the title. The first chapter is a dusting of background, theory, and terminology, closing with a walk-through of the installation process for Vagrant. By the second page of the second chapter he has you running your first VM, a stock Ubuntu image.

The process takes longer than that sounds because of the way Vagrant simplifies OS installation by providing a set of pre-built minimal installation images called "Boxes." When you first create a new VM, you will need a network connection so that the initialization process can download the initial box. Although the boxes are small for OS images, they are nonetheless complete bootable disk images for the target operating system.

The next three chapters cover provisioning a real system on top of the base image (using scripts, Puppet, or Chef), network variations, and creating complex system simulations with multiple virtual machines. The coverage is brief but clear and sufficient. Hashimoto doesn't try to fill in every detail. He indicates that the reader will need to be comfortable with an editor or with basic Ruby syntax and moves on. The examples are clear, not too cluttered, and, remarkably, not at all contrived.

The last two chapters introduce the two major ways of extending Vagrant. The first is by creating custom "Boxes." These are just VM images (see above) with a little metadata to help Vagrant manage them, but boxes provide a means for a developer or tester to throw away a polluted environment and restore to a well-known start state quickly and reliably. New boxes can be pulled from remote Web sites and are cached locally.

Hashimoto discusses plugins last. In writing Vagrant, he has used the standard mechanisms offered by Ruby to provide hooks for extending the existing behavior, both adding new commands and altering the behavior of existing ones.

Often "up and running" style books move so fast in an attempt to show everything that a tool can do that they leave the reader without a real working knowledge of the main task. Hashimoto has written about a tool with a specific purpose and has focused on showing the new user how to do that job easily. He also manages to provide hints for the possibilities for extensions.

Vagrant is a tool that helps manage desktop virtual machines. If you're a software developer for complex Web services, you're going to want to at least look at Vagrant, and *Vagrant: Up and Running* is a really good place to start. Mitchell Hashimoto is himself the creator of Vagrant; not all software developers can also write for humans, but Hashimoto is an effective advocate and instructor for his tool.

Git Pocket Guide: A Working Introduction

Richard E. Silverman
O'Reilly Media, 2013. 215 pp.
ISBN-10: 1449325866
Reviewed by Mark Lamourine

I've been a fan of the O'Reilly pocket references for quite a while. Especially when learning some new programming language or tool, I find that the pocket reference is a quick way to get what I need without the narrative of a tutorial or the depth of a traditional user guide or reference text.

The Git Pocket Guide uses the same physical format as the pocket references, but the format is task-oriented instead of the more typical feature list. This makes sense for Git as its purpose is to help manage the process of collaborative software development.

I like the way Silverman interlaces usage explanation, examples, and theory. Git takes a significantly different approach to implementing source code revision control than previous tools, such as CVS and Subversion (though more similar to Mercurial and Bazaar). Each chapter in the pocket guide provides a brief view into how Git works when performing the task under discussion. Unlike many other tools, understanding how Git works helps a lot in understanding how to use it.

The physical and structural formats create a portable and utilitarian book that should be enough for most people who are already familiar with the fundamentals of source code control to get started and do most tasks with Git.

Puppet Types and Providers: Extending Puppet with Ruby

Dan Bode and Nan Liu
O'Reilly Media, 2013. 80 pp.
ISBN: 978-1-4493-3932-6
Reviewed by Mark Lamourine

O'Reilly has recently begun publishing a series of thin short-topic books, and Puppet Types and Providers is one of that line. The authors focus on a single advanced aspect of working with Puppet: extending Puppet by adding new managed resources. This book was timely for me, and I just wish there was documentation this good for the rest of Puppet.

At 80 pages, the book does not have much room for introduction, but Bode and Liu do manage to cover enough of the Puppet internals so that the purpose and use of the Types and Providers mechanisms is clear. Then they get right to the meat.

Briefly, Puppet models target systems by allowing the user to define a set of resources that should exist on those systems. The Types mechanism provides means for Puppet developers to create abstract definitions for new resources. Providers are the concrete implementation backend for the abstract Type definitions. Especially interesting to me is the explanation of the "suitability" mechanism that Puppet uses to decide which provider to use to execute the resource requirements.

The book only has four chapters. Following an introduction to Puppet's resource model, the exposition of Puppet types and providers each takes one of the remaining chapters. The final chapter touches on several advanced features of Puppet resource management using the type/provider mechanism. Before finding this book , I looked for quite a while for documentation that clearly demonstrates how to define new Puppet resources. I'm done looking.

Statement of Ownership, Management, and Circulation, 10/1/13

Title: ;login: Pub. No. 0008-334. Frequency: Bimonthly. Number of issues published annually: 6. Subscription price $90.

Office of publication: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

Headquarters of General Business Office of Publisher: Same. Publisher: Same.

Editor: Rik Farrow; Managing Editor: Rikki Endsley, located at office of publication.

Owner: USENIX Association. Mailing address: As above.

Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, or other securities: None.

The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes have not changed during the preceding 12 months.

Extent and Nature of Circulation			Average No. Copies Each Issue During Preceding 12 Months	No. Copies of Single Issue (August 2013) Published Nearest to Filing Date
a. Total Number of Copies			3479	3100
b. Paid Circulation	(1)	Outside-County Mail Subscriptions	1723	1667
	(2)	In-County Subscriptions	0	0
	(3)	Other Non-USPS Paid Distribution	1010	987
	(4)	Other Classes	0	0
c. Total Paid Circulation			2733	2654
d. Free Distribution By Mail	(1)	Outside-County	0	0
	(2)	In-County	0	0
	(3)	Other Classes Mailed Through the USPS	64	75
	(4)	Free Distribution Outside the Mail	490	219
e. Total Free Distribution			554	294
f. Total Distribution			3287	2948
g. Copies not distributed			192	152
h. Total			3479	3100
i. Percent Paid			83%	90%
Paid Electronic Copies			417	432
Total Paid Print Copies			3150	3086
Total Print Distribution			3703	3380
Percent Paid (Both Print and Electronic Copies)			85%	91%

I certify that the statements made by me above are correct and complete.
Anne Dickison, Co-Executive Director 10/1/13

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

Free subscription to *;login:*, theAssociation'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 this 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/membership/specialdisc.html.

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.

2014 Election for the USENIX Board of Directors

Anne Dickison and Casey Henderson, USENIX Co-Executive Directors

The biennial election for officers and directors of the Association will be held in the spring of 2014. A report from the Nominating Committee will be emailed to USENIX members and posted to the USENIX Web site in December 2013 and will be published in the February 2014 issue of *;login:*.

Nominations from the membership are open until January 6, 2014. To nominate an individual, send a written statement of nomination signed by at least five (5) members in good standing, or five separately signed nominations for the same person, to the Executive Directors at the Association offices, to be received by noon PST, January 6, 2014. Please prepare a plain-text Candidate's Statement and send both the statement and a 300 dpi photograph to production@usenix.org, to be included in the ballots.

Ballots will be mailed to all paid-up members in early February 2014. Ballots must be received in the USENIX offices by March 17, 2014. The results of the election will be announced on the USENIX Web site by March 26 and will be published in the June issue of *;login:*.

The Board consists of eight directors, four of whom are "at large." The others are the president, vice president, secretary, and treasurer. The balloting is preferential: those candidates with the largest numbers of votes are elected. Ties in elections for directors shall result in run-off elections, the results of which shall be determined by a majority of the votes cast. Newly elected directors will take office at the conclusion of the first regularly scheduled board meeting following the election, or on July 1, 2014, whichever comes earlier.

USA Team Wins Big at 2012 International Olympiad in Informatics

Brian C. Dean, Director, USA Computing Olympiad

G'day all! Scenic, sunny Brisbane, Australia was the location this year for the International Olympiad in Informatics (IOI), the world's most prestigious computing competition at the high-school level, involving teams from 80 countries. This year, team USA consisted of:

- Joshua Brakensiek (Junior, Homeschooled, AZ)
- Steven Hao (Junior, Lynbrook High School, CA)
- Johnny Ho (Senior, Lynbrook High School, CA)
- Scott Wu (Sophomore, Baton Rouge Magnet High School, LA)

Continuing a tradition of strong performance at past IOIs, our showing at the 2013 IOI was quite good, with two gold medals (Brakensiek, Wu) and two silver medals (Hao, Ho), placing us among the top countries in attendance. Hosted at the University of Queensland, the week-long event included two full days of competition as well as fun excursions to the Sunshine Coast and the Australia Zoo, giving us time to interact with like-minded peers from all over the world, as well as a few Kangaroos and Koalas. On the technical side, the competition was of the highest quality, including some extremely challenging and creative algorithmic tasks. My favorite task required students to design algorithms that could quickly classify hundreds of works of art by style (e.g., neoplastic modern art, impressionist landscapes, or expressionist action paintings). It was an amazing week, and one of the best IOIs ever in my memory.

Every year, the IOI team is selected and trained by the USA Computing Olympiad (USACO). One of several major scientific Olympiads in the States, the USACO (usaco.org) supports pre-college computing in the USA and worldwide through free on-line educational material and monthly programming competitions at several levels throughout the academic year. Each summer, the USACO invites the top students in the USA to an intensive academic "training camp", held at Clemson University, where they receive advanced instruction and contend for placement on the USA team to attend the IOI. This summer, thanks to our generous sponsors such as USENIX, we were able to scale up the size of our summer camp by 50%, inviting the top 24 students in the country. Highlights of the camp experience included an excursion to Oak Ridge National Labs (home of some of the world's fastest supercomputers), a lecture and programing challenge on machine learning from distinguished alum Percy Liang (now a professor at Stanford), a game contest where students watched their programs compete head-to-head in an animated tournament, and an optimization challenge where student teams computed traveling salesman tours of the stars in the night sky, visualized on Clemson's huge planetarium dome.

The 2013 USA IOI team: Joshua Brakensiek, Johnny Ho, Steven Hao, and Scott Wu. *Photo courtesy Brian C. Dean.*

The USACO depends on a small but dedicated volunteer staff of amazingly-talented individuals, many of them former IOI champions themselves. Our staff this year includes Mark Gordon (graduate student, University of Michigan, also our deputy team leader at the IOI), Jacob Steinhardt (graduate student, Stanford), Neil Wu (undergraduate, Harvard), Nathan Pinsker (undergraduate, MIT), Dr. Richard Peng (Instructor of Applied Mathematics, MIT), and Dr. Eric Price (soon to be Professor at the University of Texas, Austin). I am exceedingly grateful to these individuals for their hard work throughout the year in organizing USACO activities, and also to our sponsors, for making our programs possible. Sponsorship by USENIX of the USACO plays a truly crucial role in maintaining the high quality of our country's computing talent.

Thanks to Our Volunteers

Anne Dickison and Casey Henderson, USENIX Co-Executive Directors

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, as well as contributing to this magazine. We are most grateful to you all. We would like to make special mention of some people who made particularly significant contributions in 2013.

Program Chairs

Keith A. Smith and Yuanyuan Zhou: 11th USENIX Conference on File and Storage Technologies (FAST '13)

Alexandra Meliou and Val Tannen: 5th USENIX Workshop on the Theory and Practice of Provenance (TaPP '13)

Nick Feamster and Jeff Mogul: 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)

Petros Maniatis: 14th Workshop on Hot Topics in Operating Systems (HotOS XIV)

Chris St. Pierre: 2013 USENIX Configuration Management Summit (UCMS '13)

Emery Berger and Kim Hazelwood: 5th USENIX Workshop on Hot Topics in Parallelism (HotPar '13)

Yixin Diao (General Chair); Jie Liu and Ming Zhao (Program Co-Chairs): 8th International Workshop on Feedback Computing

Dilma Da Silva and George Porter: 5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '13)

Uwe Brinkschulte, Christian Müller-Schloer, and Mathias Pacher: 2013 Workshop on Embedded Self-Organizing Systems (ESOS '13)

Andrew Birrell and Emin Gün Sirer: 2013 USENIX Annual Technical Conference (USENIX ATC '13)

Jeffrey Kephart (General Chair); Calton Pu and Xiaoyun Zhu (Program Co-Chairs): 10th International Conference on Autonomic Computing (ICAC '13)

Ajay Gulati: 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '13)

Nicole Forsgren Velasquez and Carolyn Rowland: 2013 USENIX Women in Advanced Computing Summit (WiAC '13)

Cristian Cadar and Jeff Foster: 5th Workshop on Hot Topics in Software Upgrades (HotSWUp '13)

Sam King: 22nd USENIX Security Symposium (USENIX Security '13)

Walter Mebane and Dan S. Wallach: 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE '13); also Editors-in-Chief of the *USENIX Journal of Election Technology and Systems (JETS)*

Chris Kanich and Micah Sherr: 6th Workshop on Cyber Security Experimentation and Test (CSET '13)

Jed Crandall and Joss Wright: 3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI '13)

Kevin Fu, Darren Lacey, and Zachary Peterson: 2013 USENIX Workshop on Health Information Technologies (HealthTech '13)

Vern Paxson: 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '13)

Matt Blaze: 2013 USENIX Summit on Hot Topics in Security (HotSec '13)

Jon Oberheide and William Robertson: 7th USENIX Workshop on Offensive Technologies (WOOT '13)

Narayan Desai and Kent Skaar: 27th Large Installation System Administration Conference (LISA '13)

Kyrre Begnum: 2013 USENIX Summit for Educators in System Administration (SESA '13)

Invited Talks/Special Track Chairs

Nitin Agrawal: Poster Session Coordinator at FAST '13

Joseph Tucek: Work-in-Progress Reports (WiPs) Coordinator at FAST '13

John Strunk: Tutorial Chair at FAST '13

Matthew Caesar: Poster/Demo Program Chair at NSDI '13

Michael Bailey (Chair), Elie Bursztein, Wenke Lee, and Stefan Savage: Invited Talks Committee at USENIX Security '13

William Enck: Poster Session Coordinator at USENIX Security '13

Nikita Borisov: Rump Session Chair at USENIX Security '13

Ben Ransford: Poster Session Coordinator at HealthTech '13

Michael Bailey: Deputy Program Chair at HotSec '13

Nicole Forsgren Velasquez and Cory Lueninghoener: Invited Talks Coordinators at LISA '13

Lee Damon: Lightning Talks Coordinator at LISA '13

Kyrre Begnum: Workshops Coordinator at LISA '13

Chris St. Pierre: Guru Is In Coordinator at LISA '13

Marc Chiarini: Poster Session Coordinator at LISA '13

Matt Simmons: Tutorial Coordinator at LISA '13

Paul Krizak, Chris McEniry, and Adele Shakal: LISA '13 Lab Hack Space Coordinators

Other Major Contributors

John Arrasjid, David Blank-Edelman, Sasha Fedorova, Brian Noble, Niels Provos, Carolyn Rowland, Margo Seltzer, and Dan Wallach for their service on the USENIX Board of Directors

Dan Geer, Eric Allman, and Niels Provos for serving on the Audit Committee

Margo Seltzer for chairing the USENIX Board of Directors Nominating Committee

Brian Noble, John Arrasjid, Sasha Fedorova, Cory Lueninghoener, and Matt Simmons for serving on the Awards Committee

Brian Dean, Mark Gordon, Jacob Steinhardt, Neil Wu, Nathan Pinsker, Dr. Richard Peng, and Dr. Eric Price, this year's directors and coaches for the USA Computing Olympiad, co-sponsored by USENIX

Eddie Kohler for his HotCRP submissions and reviewing system

Tadayoshi Kohno for organizing the Tribute to Evi Nemeth at USENIX Security '13

Paul Vixie and the ISC for providing 9-layer ISO T-shirts in honor of Evi Nemeth at USENIX Security '13 and LISA '13

Peter Honeyman for assistance with HotSec '13

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 LISA '13

Matt Simmons, Ben Cotton, and Michele Chubirka for blogging about USENIX and LISA '13 activities

Tom Limoncelli for assistance with LISA '13 promotion

David Nolan, Andrew Mundy, and Dragos Jula for assistance with LISA '13 connectivity

Andrew Mundy for assistance with LISA '13 videography

REPORTS

Conference Reports

22nd USENIX Security Symposium (USENIX Security '13)

Washington, D.C.
August 14–16, 2013

Summarized by: Theodore Book, Sven Bugiel, Rik Farrow, Xinyang Ge, Frank Imeson, Bhushan Jain, Rahul Pandita, John Scire, Gang Wang, and Ziming Zhao-

Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Sam King, the Program Chair, told us that there were 277 submissions to the 22nd Security Symposium, and that 44 had been accepted. After thanking the program committee members, Sam suggested that we not miss the rump session on Wednesday night (which turned out to be both a lot of fun and interesting).

The Best Paper award went to "Control Flow Integrity for COTS Binaries," by Mingwei Zhang and R. Sekar (Stony Brook University). The Best Student Paper award was presented to "Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation," by Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh V. Tripunitara (University of Waterloo). Finally, Sam presented Crispin Cowen with the Test of Time award for Stackguard, a mechanism that guards against stack overflows and that Crispin led the development of more than ten years ago.

Wednesday Keynote Address

Summarized by Rik Farrow (rik@usenix.org)

Dr. Felten Goes To Washington: Lessons from 18 Months in Government

Edward W. Felten, Director, Center for Information Technology Policy, and Professor of Computer Science and Public Affairs, Princeton University; former Chief Technologist, U.S. Federal Trade Commission

Ed Felten worked a year and a half as the Chief Technologist at the Federal Trade Commission (FTC). He explained what working with buildings full of lawyers was like, and what we can do to work more effectively with these people. There are differences in culture, and he adopted the mode of dress of Washington people. Emphasizing this point, Felten removed his suit coat, tie, and dress shirt and revealed a t-shirt showing Evi Nemeth's nine protocol layers, a more appropriate style of dress for USENIX conferences than the coat-and-tie of Washington.

Felten pointed out that Senator Ted Stevens got into trouble for describing the Internet as a series of tubes, but this was not that ridiculous, as we had talked about networks as pipes all the time. We still believe that politicians don't get it, but then they stereotype us as well. Felten displayed a picture of a kid in his parent's basement with cigarette smoke-stained PC XT. People in Washington did notice the SOPA and PIPA protests, so the people here do believe they need to pay attention to us. Still, how to meet and work with us remains an awkward problem. Felten used a photo of Elvis shaking hands with President Nixon as an example (https://en.wikipedia.org/wiki/File:Elvis-nixon.jpg); we need to be like Elvis and learn how to work with Nixon.

Felten then explained his job. The FTC missions are consumer protection and antitrust/competition (shared with DoJ) and involve civil law enforcement and investigation. Felten acted as the policy advisor to the Chairman of the FTC, as an internal technology consultant in the agency, and finally, as an ambassador to the tech community.

At this point, Felten's talk got really interesting as he explained politics using examples from set theory and algorithms. In our culture, we are obliged to pretend to agree on truth and to learn from each other, instead of using overheated rhetoric and bogus claims. But politics is not a search for truth, and this is a feature rather than a bug. Democracy is not a search for truth, but an algorithm for resolving disagreements—voting. With voting, all questions are decidable in constant time (O(1)). There is no need to decide issues based on underlying facts or coherent explanations.

Individual legislators appear to be logically inconsistent and indifferent to truth, but politicians behave that way for a reason. Felten then showed pseudocode to explain how politicians can appear inconsistent. He proposed that voters have a "utility function" that allows them to like or dislike bills, and he made assumptions: that the behavior of voters is sensible, and that their ratings on two disjoint bills is disjoint. Felten went on to show that because voters can like, or dislike, bills by differing amounts, it is possible for a combination of two disjoint bills to fail passage because the degree of dislike for one part of the bill is greater than the degree of "like" for the remainder of the bill. The result is that the outputs of democracy are not logically consistent. Felten expounded on this model, showing that if legislatures follow majority opinion, they will also be logically inconsistent and appear indifferent to the truth—because they are. He also pointed out that the problem of adding amendments to bills is NP complete.

Policy-makers need to be generalists, as they have a broad domain to cover and they can't be an expert in every area. Their goal is to make good decisions, and to do so they need to be clueful. Felten presented his ladder of cluefulness. The bottom rung is to recognize that expertise exists. The middle step is to recognize true experts, and the top step involves working effectively with experts. The top rung for experts is to work effectively with decision-makers. To do so, you learn about their knowledge and preferences and provide information to help fill in the structure of the decision space.

If you have reached the right point in your career, consider taking a sabbatical and working for the government, just as Felten and Steve Bellovin have done. Felten suggested spending at least a year so you can be productive. Felten also explained that for people starting out, there currently is no career path that leads from being a working technologist to senior advisor, and it would be good if that existed.

Dr. Felten left a lot of time for discussion. Tony Dahbura (Johns Hopkins) observed an important paradox that appears in society, that the more information becomes accessible, the more uninformed people's behavior appears to be, perhaps because they are reluctant to say "I don't know." Felten said that experience has shown that having more information has made people better decision-makers on the whole, but wouldn't go as far as Dahbura had in saying it was actually harmful. People need to have skills to use that information. People also are attempting to confirm their beliefs, and it has likely always been that way. Knowing how to turn information access into better decision-making is important.

Iulia Ion (Google) asked what people who don't have sabbaticals can do to get involved and share their own views. Felten suggested getting in touch with a policy-maker or the people on their staff and developing a contact point. The staff people who answer the phone or work in the office are there largely to work with constituents, and educating a staff member well might have a greater impact than talking directly to the decision-maker.

Greg Shannon (CMU) pointed out that some organizations have legislative affairs people. Then Greg asked what it was like to have someone listen to him. Felten had done a lot of due diligence before going into the job, and knew he had to work with the people in the FTC who knew how to work with Congress and other decision-makers. Felten said you have to socialize the idea you want to get across, and he worked early on to develop a rapport with FTC staff members. Government is designed to make things hard to do, the checks and balances put there to prevent abuse. Government is closer to university politics than you might think, quipped Felten.

William McAlter (Johns Hopkins Applied Physics Lab) asked where Felten learned about working in government. Felten said he learned about this through his struggle with the DMCA and how it affected his research, and later through being an advisor on the Microsoft antitrust case. Felten said it is something you have to learn over time.

Joe Kiniry (Technical University of Denmark) said that, having worked in both Europe and America, he had discovered some differences. For example, in Denmark, there is not a single legislator educated in STEM. In America, that tastes different. Felten replied that having politicians trained in sciences is a good thing; for example, there's a New Jersey senator who was trained in physics and actually understands statistics. In the House, on the other hand, STEM education is quite rare, which becomes an issue. Part of the issue is the career gap, and another is the belief that knowledge of technology disqualifies you from participating in that policy discussion.

Bill Simpson thanked Felten for a great call to arms. He also suggested getting involved in campaigns, as he has done, by providing technical support. Simpson pointed out that of those people you are participating in campaigns with, about half of them will become staffers. Simpson said he has been doing this since the mid-'70s, and now visits people he knows in congressional offices when he visits Washington. Felten agreed that this is excellent advice, and went further by saying that campaigns have become much more analytical and data driven, so there is now a greater need for technical support, to apply your expertise to campaigns.

Chris Watjic (Georgetown) wondered how to help politicians identify quacks. Felten suggested helping people recognize what type of credentials represent expertise, such as being a long-time member of the IETF (like Bill Simpson), or being a program chair or program committee member. Unfortunately, sometimes credibility comes from a person who works for a company that has a stake in the outcome of a decision.

Michael Hicks (University of Maryland) asked whether there is a way that researchers could do their jobs better to help with the political process. Felten said that we currently focus on building knowledge brick-by- brick, but sometimes we need to choose our projects differently. Also, we need to examine how we decide to publicize our findings, which could be as simple as emailing a contact about your research.

There was much more discussion, and additional points that Felten provided in his well-received and prepared talk. I suggest that you watch the video or listen to the audio on the USENIX Web site: https://www.usenix.org/conference/usenixsecurity13/dr-felten-goes-washington-lessons-18-months-government.

Network Security

Summarized by Gang Wang (gangw@cs.ucsb.edu)

Greystar: Fast and Accurate Detection of SMS Spam Numbers in Large Cellular Networks Using Gray Phone Space

Nan Jiang, University of Minnesota; Yu Jin and Ann Skudlark, AT&T Labs; Zhi-Li Zhang, University of Minnesota

Yu Jin talked about Greystar, their system for detecting SMS spam in cellular networks. The authors' key assumption is that spammers randomly select target phone numbers from a finite phone number space. So they will inevitably send messages to numbers that normal users typically would not reach: for example, those associated with laptop data cards or electricity meters. Yu called these numbers "gray" phone numbers.

Then Yu described their statistic model for SMS spammer detection based on gray phone numbers. To evaluate the system, they experimented with five-months of SMS call records from AT&T. The experiments demonstrated that they could achieve high accuracy, and also detect spammers much faster than existing crowdsourced user reports. In particular, Yu mentioned that their system, once deployed, could reduce spam volume by 75% during peak hours.

Several people asked about the possibility of using other features to improve the system, for example, messages sent per day. Yu responded that these features were complementary, and some could easily raise false alarms. Another audience member asked what would happen if attackers didn't target randomly selected phone numbers but real, valid phone numbers collected via other methods, e.g., social engineering. Yu said that, based on their real-world data, 90% of the spammers fall into their assumption. Finally, there was a question about possible collaboration of different carriers to combat SMS spam together. Yu said collaborations would be helpful, in their case, to accurately identify gray phone numbers and catch spammers; however, in practice, this type of collaboration was still hard to achieve.

Practical Comprehensive Bounds on Surreptitious Communication over DNS

Vern Paxson, University of California, Berkeley, and International Computer Science Institute; Mihai Christodorescu, Qualcomm Research; Mobin Javed, University of California, Berkeley; Josyula Rao, Reiner Sailer, Douglas Lee Schales, and Marc Ph. Stoecklin, IBM Research; Kurt Thomas, University of California, Berkeley; Wietse Venema, IBM Research; Nicholas Weaver, International Computer Science Institute and University of California, San Diego

Wietse Venema presented their work on detecting stealth communication over DNS. Today, attackers can piggyback communication in DNS queries to transmit information secretly. Wietse presented a new measurement procedure that could bound the amount of information that a domain could receive through DNS queries. The key idea is to use lossless compression. Potentially, attackers may encode information in a DNS query name, query type, query timing, or a combination of them. The authors' procedure takes all potential information vectors and investigates the upper bound of information that can be encoded in a stream of DNS queries. Using this bound, they can narrow down surreptitious communications to a small set of DNS lookups. Also, the set should be small enough for manual assessment.

A practical challenge for this procedure is how to minimize the analysis burden in the face of tens of millions DNS lookups. In the talk, Wietse showed how they pare down the volume of DNS queries by eliminating obvious benign candidates. They evaluated this procedure with a real-world data set of 230 billion DNS lookups. Their procedure had no false positives and was able to detect 59 confirmed tunnels. Wietse also pointed out that they found that 4 KB/day was a reasonable threshold, which led to an acceptable assessment burden (one to two events per week) for enterprise sites to take in practice.

One audience member asked whether they could share the data set. Wietse said they were happy to share the code and results, but the data set was from IBM and could not be shared because of company policy. Another audience member asked whether this approach would still work if DNS queries were encrypted. Wietse's reply was positive. Someone asked how they determined the thresholds in the measurement procedure. Wietse said that the tradeoff was made based on their empirical analysis of real data: a smaller threshold (4 KB) for individual clients and a larger threshold (10 KB) for extremely aggregated logs.

Let Me Answer That for You: Exploiting Broadcast Information in Cellular Networks

Nico Golde, Kevin Redon, and Jean-Pierre Seifert, Technische University Berlin and Deutsche Telekom Innovation Laboratories

Kevin Redon presented a new attack in cellular networks. Focusing on GSM, he demonstrated how attackers could hijack a mobile terminated service (e.g., phone call) and perform a denial of service attack. This attack can occur because GSM initiates the paging procedure on a broadcast medium before setting up any authentication. So attackers who are also in this network can observe the paging requests of other phones (victims) and send a fake paging response on behalf of the victim. If the attacker responds faster than the victim, the GSM network will accept the fake response and ignore the victim's response. After these replies, GSM will set up service authentication (which will fail) and the victim's service will be dropped.

Kevin demonstrated the feasibility of this attack using freely modifiable software and hardware for GSM networks. Other standards, such as UMTS or LTE, also have the same (vulnerable) paging procedure, which is worth noting. At the end of the talk, Kevin showed a list of possible countermeasures, using A5/3 encryption to prevent hijacking, for example, or performing authentication before paging procedure, etc. Kevin said they notified the respective standards organizations about this problem but have had no immediate reaction from them so far.

Video about the attack can be found here: https://www.youtube.com/watch?v=oep3zpY6cvE, https://www.youtube.com/watch?v=4umb2P-93BQ.

One audience member asked which countermeasure is actually deployable in practice. Kevin said most countermeasures are about protocol modification, which requires efforts from standards organizations. At the very least, we could adopt the more secure A5/3 to mitigate the threat. A follow-up question was whether they tested any proposed countermeasures using their testbed. Kevin said they empirically tested a few, but not all of them. Another audience member asked whether the cellular tower could notice the presence of this attack based on the duplicated paging responses. Kevin said the cellular tower could detect that there were two phones sending responses, but could not tell which one was the legitimate one.

Potpourri

Summarized by Ziming Zhao (zzhao30@asu.edu)

Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations

Istvan Haller and Asia Slowinska, VU University Amsterdam; Matthias Neugschwandtner, Vienna University of Technology; Herbert Bos, VU University Amsterdam

Istvan started his presentation by explaining that buffer overflows are still among the top three threats after 40 years of research. He then provided context about state-of-the-art automated testing approaches by explaining static analysis and symbolic execution. Static analysis is difficult to make path-sensitive and inter-procedural, and it generates many false positive and negatives. Even though symbolic execution could achieve significant code coverage, the exponential number of possible paths means it is not practical in many cases.

By showing a piece of buggy code from the Nginx Web server, Istvan concluded that complete code coverage cannot even guarantee triggering a bug. To address these issues, Istvan and his co-authors tried to narrow down the scope of their research problem. Instead of pursuing complete coverage of paths, they focused on high-priority code fragments, especially the code that accesses an array in a loop.

They proposed to first identify and rank loops based on their bug probability, calculated from features such as whether the loop has a pointer dereference. Using taint tracking, they were then able to identify the variables that may influence potential buggy loops. Finally, they performed symbolic execution only on these identified variables, which reduces the test space tremendously.

To explain their symbolic execution approach, Istvan first laid out the basics of symbolic execution followed by some traditional search strategies, such as depth first search and code coverage. They proposed using a value coverage search strategy, which showed incredible performance in terms of search time. In conclusion, Istvan showed that their implemented tool, Dowser, could detect bugs in less than a minute for some programs that previously had required more than eight hours analysis.

Someone asked how their dynamic analysis was guaranteed to find the code that modifies pointers. Istvan answered that the learning process was important; more time spent on learning would increase the quality. Another attendee asked how the results of value coverage were searched without source code. Istvan replied that the only part of their analysis using source code was the static analysis to find loops. Someone asked which semantic engines was their work based on. Istvan replied they used some standard semantics engines that have been out for years.

MetaSymploit: Day-One Defense Against Script-Based Attacks with Security-Enhanced Symbolic Analysis

Ruowen Wang, Peng Ning, North Carolina State University; Tao Xie, University of Illinois at Urbana-Champagne; Quan Chen, North Carolina State University

Ruowen Wang started his presentation by introducing Metasploit, a Ruby-based penetration framework that contains more than 1,000 attack scripts. The typical mechanism that a Metasploit script uses has four steps: it (1) probes a vulnerable target, (2) generates an attack payload dynamically based on the probe results, (3) sends that payload to the victim, and (4) triggers the vulnerability and compromises the target.

He then showed a number of Internet news articles about hackers using Metasploit to attack production systems; Metasploit as a powerful penetration tool has turned into a real-world weapon. Ruowen and his co-authors have proposed an effective technique to defend against attacks launched by Metasploit. He explained that their approach does not require a vulnerable applications and testing environment, but only uses security-enhanced symbolic analysis to generate IDS signatures.

Ruowen presented the architecture of their tool, MetaSymploit. MetaSymploit symbolically executes attack scripts collected from Metasploit and captures fine-grained attack behaviors and conditions. By using both symbolic values and concrete values in the generated payload from MetaSymploit, they were able to extract signature patterns for specific attack payloads.

To implement their idea, Ruowen presented their efforts to develop a symbolic execution engine for Ruby 1.9.3. They have integrated their work into Metasploit 4.4. Based on their evaluation, their tool could generate snort roles for 548 attack scripts. The performance required less than one minute for each script, wihch is impressive considering that symbolic execution was adopted.

An attendee asked whether Ruowen had considered combining the generated rules. Ruowen replied that they are looking into some work on aggregating rules with regular expressions. Session chair David Wagner asked how hard it is for the bad guys to defend against this work. Ruowen said it is possible for bad guys to defend against their technique, but they face challenges. Shuo Chen (Microsoft Research) asked whether the input size of the symbolic execution introduced any performance issues. Ruowen replied that it was not an issue in their study.

Towards Automatic Software Lineage Inference

Jiyong Jang, Maverick Woo, and David Brumley, Carnegie Mellon University

Jiyong Jang explained the motivations for software lineage inference, which is to recover the lineage given a set of program binaries. Software lineage inference could provide information in many security scenarios, such as malware triage and software vulnerability tracking. Even though there are abundant analyses of software history and lineage, how to infer software lineage from binaries automatically is still an open question.

To address this problem, Jiyong presented a list of software features that could be utilized to infer a temporal ordering and evolutionary relationships among binaries. He also explained some features were chosen based on the common understanding that program size and complexity tend to increase rather than decrease as new revisions are released.

To measure the difference between the feature sets from binaries, Jiyong presented several techniques that include symmetric distance, dice coefficient distance, Jaccard distance, Jaccard containment distance, and weighted symmetric distance. Jiyong then showed that the lineage inference algorithm they proposed performed similarly regardless of the distance metrics, with Jaccard containment distance being the exception.

To evaluate their work, Jiyong presented some lineage examples from real-world binaries compared with ground truth generated from source code. Jiyong focused on one example in which the automatically inferred lineage differed from the ground truth. Jiyong explained that a deeper manual analysis revealed that a version of the software was reverted to version 1 after several generations instead of evolving from the previous version. This was the root cause of the difference, and their automatically inferred results were accurate and able to identify this change.

Sumam Jana (UT Austin) asked whether their work is based on source code or binary. Jiyong replied their work only needs binaries. One attendee asked about how they handled obfuscated malware. Jiyong replied they had considered a lot of metrics, including some dynamic features, and had combined them with other features to achieve better accuracy for malware. Someone from Maryland asked about the particular challenges involved in extracting lineage relationships from binaries since there is already work doing the same thing for source code. Jiyong said working on binaries required much more careful feature selection.

Mobile Security I

Summarized by John Scire (jscire@stevens.edu)

Securing Embedded User Interfaces: Android and Beyond

Franziska Roesner and Tadayoshi Kohno, University of Washington

Franziska described the motivation for their work on embedded user interfaces. Currently, Web browsers have a simplistic and mostly secure way of embedding third-party material into a Web site using iframes, which provide secure isolation between UI elements; however, Android does not have any way to do cross-application embedding securely. What currently exists in Android is the embedding of ads in an application, but this is done using third-party ad libraries. She gave a great example to demonstrate a type of attack that exists with these ad libraries on current stock Android, where an embedded ad could change all of the other child UI elements in an application. She went on to describe some of the previous work related to the embedded UI in Android, but these only involved approaches specifically tailored to these ad libraries. The approach that her team took was creating a modified version of Android that supports secure cross-application embedding, which they call LayerCake.

Franziska provided some background knowledge about how Android applications work so as to understand how their modification works. An Android application consists of one or more elements that are known as Activities and within each Activity there is a tree of UI elements known as Views. The modification itself, as described by Franziska, involves three components. The first is the separation of processes, which essentially works similarly to iframes. They created a new View called EmbeddedActivityView that will display the embedded content. This new addition allows the parent and child elements to be isolated from one another, while still having communication between them. The second component is to use separate windows for each of these Embedded Activity Views. This is because their first component, creating new Views, still allows for UI elements to grab data passing through the layout tree. The third component involves various other additions to handle other security concerns discussed in the paper.

The evaluation of LayerCake involved, in total, more than 2,500 changes that included fundamental changes to the ActivityManager and WindowManager. In the applications they tested, higher load times were required to load all of the embedded activities. The parent activity load times, on the other hand, were unaffected. Because each Activity is in its own window, the Android WindowManager has to be involved to switch focus based on user input. This additional indirection, however, had little impact on the application. For instructions on how to download and flash LayerCake onto an Android device, go to http://layercake.cs.washington.edu.

Will Enck (NC State) asked about how this modified Android would handle software dependencies with embedded UI elements in an application. Franziska replied that there was not one real answer, but she provided some approaches, including installing the dependencies at the Android store. Paul Pierce (UC Berkeley) asked about having any plans with Google to integrate this into stock Android. Franziska replied that she had not talked to Google about this yet, but would love to.

Automatic Mediation of Privacy-Sensitive Resource Access in Smartphone Applications

Benjamin Livshits and Jaeyeon Jung, Microsoft Research

Ben began his talk by providing an overview of permissions in mobile applications. Permissions mainly go under two categories, installation-time and runtime, these names describing the point at which they are shown and asked for. Installation-time permissions were not enough, however, because users simply click "Accept" and continue using the application without really knowing what they are consenting to. Although this may have implications on iOS and Android, the rest of the talk focused on location data permissions on the Windows Phone platform.

Ben explained an MS guideline document that has various criteria for properly obtaining a user's permission, which in the scenario of location data requires having some kind of prompt telling the user that an application wants to use her location. By looking at how various example applications implemented prompts, Ben and his team were able to come up with a static analysis approach using a Control Flow Graph to locate missing prompts for resources and put them in when they were actually missing. They developed two different methods to do this. The first was the Dominator-Based method, where the prompt would be placed at the dominating node for a particular access request node. Ben said that this method, although extremely fast, prompts the user long before the actual request, which was something that they wanted to avoid. The other method was Backward Placement, which works backwards through the graph, starting at the resource access and putting the prompt at nodes prior to these accesses. The problem with this approach is that you could have multiple placements of prompts for the same access.

The authors evaluated 100 applications with an average size of 7.3 MB and an average of two location accesses per application. The Dominator method was faster than Backward Placement and was also much more successful in terms of properly inserting missing prompts in applications. Taken together, these approaches were 91% successful and, for unique resource accesses, 95% successful in correctly placing missing prompts.

Rik Farrow asked why this approach wasn't just put into the OS itself. Ben said that this not only required a lot of "soul searching," but also a bit more than simply placing it into the OS. He added that they also want to allow the developer to have some control. Someone asked about checking what the actual prompt says if it does exist within the application. Ben replied that they do not have any further analysis on the actual prompt text. The questioner said that you could build this into the OS by having mandatory text and then optional text with a particular prompt. Ben said that this was not impossible to do.

Flexible and Fine-Grained Mandatory Access Control on Android for Diverse Security and Privacy Policies

Sven Bugiel, Saarland University; Stephan Heuser, Fraunhofer SIT; Ahmad-Reza Sadeghi, Technische Universität Darmstadt and Center for Advanced Security Research Darmstadt

Sven started by briefly describing the current state of Android security, which has proven to be insufficient several times over using various attack vectors. Thus, better security mechanisms need to be in place. He introduced previous academic security extensions that have been developed, such as Saint, XManDroid, and SEAndroid. From these, Sven and his team made two key observations: (1) most of these extensions involved a form of mandatory access control that was modified to fit a specific problem and not a general fitting, and (2) access control on Android needs to be both on a user-space level and a kernel-space level. Sven mentioned a particular example of a rootkit bypassing a middleware enforcement mechanism altogether to access a particular service within Android. Using these two observations, Sven and his team came up with a general system-wide mandatory access control solution for Android called FlaskDroid.

FlaskDroid employs a policy language, SELinux to be specific, in order to perform the MAC enforcement policies. Along with this, it uses an object manager that allows processes or applications to be aware of the exact kind of data they are handling, which includes attributes such as a particular security type for that object. Examples of the language were provided as further explanation, but there are a multitude of them in the paper. In terms of the system itself, FlaskDroid uses SEAndroid at the kernel layer of Android for low-level MAC and a middleware module at the user-layer. Both of these components sit behind the API for services on Android to control enforcement and are connected to the security servers for policy queries. Sven added that the user and application developer can add policy rules specific to the settings they want that will get updated on these servers. Then, to hook the two components together, they use a Boolean mechanism whereby both the user-layer MAC and kernel-layer MAC communicate.

Because this employs the SELinux policy language, one could argue that this might weigh down FlaskDroid with an overwhelming number of rules. As it turns out, Sven and his team produced vastly fewer rules than SELinux in FlaskDroid's current setup. He also showed some use-cases pertaining to how a sample application may utilize this new MAC mechanism. One example involved a phone dialing application where the user is presented with a dial pad. The user can then turn on a phone booth context, which is a sort of mode in SELinux, that will disable the ability to leave the dial pad screen entirely. This way a person using your phone to try to dial a phone number cannot use the phone to do anything else. The paper itself has many more use cases and the source code for FlaskDroid can be viewed at flaskdroid.org.

Rik Farrow asked about the ability of malicious applications to loosen the "everything denied by default" approach of SELinux. Sven replied that the policy set by an application is only for the application and cannot interfere with access to another application. Will Enck (NC State) asked about the choice of using SELinux in the implementation due to its unmanageability. Sven responded that the choice was primarily due to wanting to merge their implementation with SELinux. Sven stated that SELinux becomes unmanageable only because of the sheer number of rules, but for smartphones it was not nearly as bad; however, Sven said they could improve this if they did in fact choose a different language.

Invited Talk

Summarized by Rahul Pandita (rpandit@ncsu.edu)

Windows 8.1 Supporting User Confidence

Crispin Cowan, Senior Program Manager, Windows Core Security, Microsoft, Inc.

Cripin started the presentation by sharing with the audience his experience of a 2010 talk where he compared Windows security with UNIX security, and humorously admitted that he was a UNIX fan prior to working at Microsoft. In retrospect, he added, Windows security was fine even then, but he pointed out that not only have attackers gotten better, but end users have become more demanding. These two factors have significantly increased the need for security in the operating system environment.

Crispin then dived deep into the features of Windows 8 directed towards boosting end-user confidence in the security of the operating system. He touched on a wide range of features, starting with hardware-based security, where he introduced Unified Extensible Firmware Interface (UEFI). UEFI is an improvement over the existing Basic Input/Output System (BIOS) to ensure that only a verified OS loader is used during boot time. This effectively addressed issues with malware that targets OS loaders. He then went into details about other security measures to ensure a safe boot of the OS in Windows 8.

Moving forward, he introduced the security features of interacting with the Windows App Store. He presented the feature called app container. An app container allows the OS to contain the effects of a rogue app installed by a user. App container also facilitates the seamless transfer of data with the OS (like opening a file) and the app by use of a mechanism Microsoft terms an authentic user gesture (AUG). The security principle behind the functioning of AUG is that the AUG can only be initiated by a user and not by an App. This was followed by a series of demos of AUG, mostly involving opening and storing a file within an app.

Crispin also presented the concept of a kill bit (reminds me of a kill switch) in apps. Having the kill bit in place allows Microsoft to remove a rogue app from all the devices remotely. He assured us that such a capability is used sparingly and after careful evaluation of the app that needs to be removed. He also explained that every app that is installed on Windows 8 has to be digitally signed by the developer and has to be installed only through the Windows App Store.

Among other features, he talked about modernized access control. In particular, he presented new sign-in options in Windows 8, including pin, passwords, picture passwords, access cards, and even biometric verification support. He proceeded to show a demo of the picture passwords but could not show it in action due to screen resolution issues of his Windows 8 device when connected to the projector for the talk. He concluded his talk by reiterating some of the core security features of Windows 8.

Felix "FX" Lindner (Recurity Labs) asked why Microsoft delegated the task of issuing and managing certificates for the OS Loader in UEFI to a third party. Crispin responded that certificate authorities (CA) were a well established business and outside the scope of Microsoft's business interests. Furthermore, he said that existing certificate authorities were doing a great job, and thus Microsoft did not feel the need to manage certificates on their own.

Someone followed up by asking, what if the CA itself was compromised? Crispin said that there was a kill-bit built right into the UEFI module to remotely disable it.

Two people asked about the kill-bit and expressed their concerns about abusing them. Crispin addressed their concerns by assuring them that Microsoft carefully weighs its options before using the kill-bit and that extra carefulness is required because abusing kill-bits also has legal implications.

Another attendee followed up by asking, what if a security researcher wanted to keep a malicious app for experimenting on it? Crispin clarified that the kill-bit was mandatory and not optional and so, if exercised by Microsoft, the malicious application had to go. He hinted, however, that there were some indirect workarounds if someone wanted to keep a malicious app.

Session chair Wenke Lee(Georgia Tech) asked whether the Surface RT—the first device that ships with Windows 8—is locked into the Windows App Store. Crispin affirmed this. Lee further inquired how difficult it is, given the safety features of the Windows 8, for students to write and install their own Apps. Crispin humorously responded that "students might have to jump some hoops to do that."

Applied Crypto I

Summarized by Bhushan Jain (bpjain@cs.stonybrook.edu)

Proactively Accountable Anonymous Messaging in Verdict

Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford, Yale University

Henry Corrigan-Gibbs outlined the motivation for their work on an anonymity system called Verdict by presenting a scenario: an activist in a country X learns about a prime minister's stash of stolen money and wants to publish this information anonymously before the elections next day. Henry then took us through the options available to the activist based on existing systems and pointed out possible attacks to expose the activist or delay message posting. An onion routing solution can be broken by a state-owned ISP using a first-last correlation attack. Henry then introduced dining cryptographers networks (DC-nets), anonymous communication networks resistant to traffic analysis in which a group of people contribute an equal length message to derive one single anonymous message at the end of the protocol. Dissent is a practical implementation of DC-nets; however, the prime minister's supporters can infiltrate the group and cause denial of service attacks on the Dissent system. The shuffle protocol used by Dissent to assign blame to the disruptor takes time and so the PM's supporters can postpone the posting of an anonymous message until after the elections are over. If the disruptors can control 10% of the nodes involved in the protocol, they can block communication for a day or more.

Verdict is a system derived from Dissent to leverage the traffic analysis resistance and scalability of Dissent but with lower blame cost. The main idea is that the group members must prove that the message they are sending is correctly formed. Thus, Verdict identifies the disruptors before they launch the denial of service attack.

Henry then took us through design challenges and optimizations to make the system fast. Verdict resists traffic analysis attacks by having each client transmit an equal length cryptographically indistinguishable message per round. In order to make the sender's transmission indistinguishable, every other client sends a dummy message encrypted using an ElGamal-like scheme while the sender sends the original message encrypted in the same format. In order to prove that their transmissions are well formed, the clients attach non-interactive zero-knowledge (NIZK) proofs of knowledge to their ciphertexts. He explained optimizations to improve performance in case of long messages, lazy proof verification, and a hybrid Dissent+Verdict DC-net.

The fastest implementation of Verdict provides a 5.6x speedup over existing systems and the hybrid Dissent+Verdict implementation gives 138x speedup. In a 1024-node cluster, the lazy Verdict optimization reduces messaging latency by 2.3x over pure Verdict, and the hybrid version reduces latency by 27x. The pure Verdict version can reduce the cost of finding disruptors from Dissent by about 200x.

When asked why not use the provable shuffle anonymity system instead of Dissent to relay messages, Henry said that Verdict achieves better efficiency for messages of varying length or multiple rounds over using provable shuffle. Someone asked whether the provable shuffle to assign slots can be replaced by a rotation. Henry said that it would work but may not be any faster. When asked if the hybrid version may take more time due to disruptions, Henry agreed that it takes time to switch to Verdict from Dissent in case of disruptions and that the hybrid version trades off the performance in the general case with the performance during disruptions.

ZQL: A Compiler for Privacy-Preserving Data Processing

Cédric Fournet, Markulf Kohlweiss, and George Danezis, Microsoft Research; Zhengqin Luo, MSR-INRIA Joint Centre

Cédric Fournet presented their work on a compiler for data processing with strong privacy guarantees. He started by explaining the need for privacy-preserving data processing using examples of smart meters and pay-how-you-drive insurance. The main argument is that the service provider doesn't need to know all the details of usage as long as the provider is getting paid the correct amount. The existing cryptographic solutions need intervention from security experts every time the policy or query is changed. To solve this problem, Cédric introduced ZQL, a high-level language for querying data together with its query compiler that synthesizes cryptographic protocols from a source definition to generate code that can run on various platforms.

ZQL supports a subset of F# language and iterators on data tables. ZQL can compute math functions, exponentiation, and table lookup operations while operating on secrets. ZQL uses a combination of Pedersen commitments, NZIK arguments, and CL-signatures for cryptographic implementation. One limitation of ZQL is that the intermediate result structure has to be public even though contents in that structure are private. The ZQL compiler takes the data specification and query as input and generates queries for the parties involved to be used in a cryptographic protocol. A F# or C generator then consumes these queries and outputs reference implementation in F# or C.

They extended ZQL to support cryptographic primitives like long integer, exponents, hashes, signatures, and commitments. Now, ZQL generates an extended query from source query using a compositional shared translation by inserting commitments, openings, and proof assertions. The extended query is subject to code specialization to generate a NIZK proof of knowledge. They also use the extended query to generate a simulator to reason about privacy and an extractor to reason about soundness.

Cédric then demonstrated the system for two sample computations and their verification. He showed how a verifier can verify that the value x+y is computed correctly. He also showed how the protocol works in the case of a pay-how-you-drive query. The cryptographic evidence is linear in size as compared to the computation. The verification proof does not contain any information about the input but provides computational integrity. The system was evaluated using RSA 1024, RSA 2048, and pairing-based crypto. The proof size is a few KB for the test cases.

When asked about the different tradeoffs for one of the related works, Pinocchio, Cédric mentioned that Pinocchio proofs are constant size and the verifier computations are small but the prover has to do more work. Pinocchio may be preferred for computation-intensive processing for a limited amount of data while ZQL will do better for large amounts of data processing.

Table lookups that are very important for ZQL cannot be done using Pinocchio; however, Cédric et al. are looking at how to combine the two. The code will be available soon.

DupLESS: Server-Aided Encryption for Deduplicated Storage

Mihir Bellare and Sriram Keelveedhi, University of California, San Diego; Thomas Ristenpart, University of Wisconsin-Madison

Sriram Keelveedhi presented a system called DupLESS, a server-aided encryption system for deduplicated storage. Deduplication saves storage resources by avoiding storing duplicate copies of the same file. Their goal is to securely deduplicate in the presence of an untrusted storage service and to provide client compromise resilience. DupLESS trades off the storage savings and performance efficiency of plaintext dedup with increased security and compromise resilience for client files. He explained how existing solutions either do not allow deduplication or are not resilient to client compromise. Even the convergent encryption solution that achieves deduplication and compromise resilience is vulnerable to brute-force attack by the storage provider to recover the original file.

DupLESS uses server-aided encryption by leveraging a key server that helps clients encrypt their files. Every client sends the key server a hash of the file, and the key server computes the key to be used to encrypt the file using a PRF on this hash value. All clients encrypt their files with the same key K and send this encrypted file along with the encryption of the key K under their own key. The first file is deduplicated as it is the same for all clients. The second file is small enough that even though it is not deduplicated, the overhead isn't much. This scheme falls short when a strong adversary can compromise the key server and leak the key K used to encrypt all copies of that file. They implemented an oblivious PRF protocol between the key server and the client to defend against this attacker. This protocol is optimized to use sessions between the client and the key server, and the actual OPRF query takes a single round as authentication is done only during session establishment. On evaluating this protocol on EC2, they observed that the protocol performance was close to round trip time for the optimized version.

Sriram then explained the details of the DupLESS system design, which uses a storage service that provides a set of APIs to manipulate files. He then took us through the translation of a storage put query to steps for DupLESS. Put and get were the most expensive operations for DupLESS. A put operation takes 16% extra time to upload a file and increases the size by about 10%. DupLESS costs 4.4% extra space as compared to plaintext deduplication. In the future, DupLESS may support keyword search, complex file systems, and heuristics on which files to select for deduplication.

Indranil Banerjee (Qualcomm) asked what secure means in the context of deduplication. Sriram replied that security implies semantic security and no information leakage about the data. A follow up question was how does deduplication increase the risk of compromising confidentiality. It is difficult to combine encryption and deduplication as seen in existing solutions, and DupLESS provides a solution to mitigate risks of attacks against these solutions. Someone asked whether a key server can do brute-force attacks on the file if the key server is compromised. Sriram replied that this attack is possible only if the key server can monitor the network traffic to get the ciphertext. Does the implementation have to take into account the backend storage provider? As long as the storage provider exposes APIs as discussed, DupLESS is seamless to the implementation behind the scenes. Zack Peterson asked why couldn't an encrypting proxy perform all the computations instead of the client. The encryption proxy becomes the natural target for the attacker, answered Sriram. With DupLESS, even if the keyserver is compromised, they at least have guarantees of a convergent encryption. David Jacobson (Qualcomm) asked whether a side channel could leak information that the file already existed on the storage server based on the time required to store the file. Sriram said that the information that is leaked is based on the location of deduplication. Deduplication on the storage provider side will force transmission of the whole file irrespective of whether the file already existed on the storage server. Someone asked why not use DTLS instead of the OPRF protocol. Sriram replied that the OPRF protocol can be derived by tweaking the DTLS protocol.

Large-Scale Systems Security I

Summarized by Gang Wang (gangw@cs.ucsb.edu)

Trafficking Fraudulent Accounts: The Role of the Underground Market in Twitter Spam and Abuse

Kurt Thomas, University of California, Berkeley, and Twitter; Damon McCoy, George Mason University; Chris Grier, University of California, Berkeley, and International Computer Science Institute; Alek Kolcz, Twitter; Vern Paxson, University of California, Berkeley, and International Computer Science Institute

Kurt Thomas presented their study on underground markets that trade fake Twitter accounts. To understand this problem, they monitored 27 account merchants over 10 months and purchased 100k fake Twitter accounts from them. Kurt said they found these merchants were using many sophisticated methods to circumvent automated account creation barriers. For example, account merchants used crowdsourcing services to solve CAPTCHAs, collected fraudulent email credentials from Hotmail and Yahoo, and also used tens of thousands of IPs (proxies, VPNs) all over the world to evade IP blacklisting.

To detect these auto-generated accounts, they developed a classifier, which looked at patterns in naming conventions and features that indicate automated accounts registration (e.g., events sequence triggered during signup and timing). With the help of Twitter, they scanned all Twitter accounts registered last year and found several million fake accounts. According to Kurt, the revenue of these account merchants are about $100,000 to $400,000. After Twitter adopted this technique, many account merchants started to go out of business. During the talk, Kurt even showed screenshots of some merchants' announcements, saying that they could no longer provide the service due to unknown changes in Twitter, which is impressive.

Someone asked about possible evasions of the proposed classifier. Kurt commented that account merchants may get around the naming features, but it was still hard for them to deal with features indicating automated account registration. Another audience member asked about the acceptable false positive rate for Twitter. Kurt said he did not know because this was confidential for Twitter. An attendee asked whether these fake accounts include compromised accounts. Kurt said the markets they focused on were mainly selling automatically registered accounts, but there are merchants who sell compromised accounts. Another person asked whether they monitored the price change over time. Kurt said that the prices in the markets they monitored were relatively flat. Someone asked whether these merchants would resell those accounts. Kurt confirmed that certain merchants did scam their customers: after selling the accounts, the merchants would try to secretly retrieve the accounts back and resell them to other customers.

Impression Fraud in Online Advertising via Pay-Per-View Networks

Kevin Springborn, Broadcast Interactive Media; Paul Barford, Broadcast Interactive Media and University of Wisconsin—Madison

Kevin Springborn talked about their measurement study on impression fraud in online advertising. In regular online advertising, advertisers place advertisements on publishers' Web sites and pay publishers based on how many users have viewed the ads (impressions). In the talk, Kevin described pay-per-view (PPV) networks that help dishonest publishers to drive traffics to their Web sites. PPV networks usually consist of compromised Web sites that render publishers' pages hidden in requested pages to users' browsers. In this way, they can generate additional, fraudulent impressions on publishers' pages. The true victims are advertisers who have to pay dishonest publishers for those impressions.

Kevin described their measurement approach. Basically, they set up three Web sites as honeypots, and then purchased traffic (addition impressions) from 34 traffic generation services who owned PPV networks. Surprisingly, they found those pay-per-view networks were rarely blocked by public blacklists and only had modest IP reuse. Additionally, there was zero user-interaction from the purchased traffic. According to Kevin, the estimated fraudulent impressions delivered by PPV networks can reach as much as 500M per day, making this a multi-hundred-million dollar business. Kevin also pointed out some possible countermeasures, such as detecting zero-sized frames and blocking known PPV hosts.

One audience member asked whether all sites in pay-per-view networks were high-quality sites. Kevin answered that the quality level may vary from service to service. Someone asked about the click-through rate of these ads. Kevin said the ads were not actually "displayed." They were usually hidden in a zero-sized frame that users cannot see. So there were no user clicks generated. Finally, someone asked about the effectiveness of the countermeasures. Kevin said the countermeasure was easy to deploy and should be effective, but many current sites did not bother to do that, because they didn't have the incentive (they were not the victims).

The Velocity of Censorship: High-Fidelity Detection of Microblog Post Deletions

Tao Zhu, Independent Researcher; David Phipps, Bowdoin College; Adam Pridgen, Rice University; Jedidiah R. Crandall, University of New Mexico; Dan S. Wallach, Rice University

Tao Zhu talked about their measurement efforts to understand censorship in Chinese microblogging sites. Their focus was Weibo, the largest microblogging site in China. Because of censorship, people's posts (i.e., tweets) on Weibo would be deleted if the content were considered to be politically sensitive. The key question Tao wanted to explore was how fast the content deletion happened and possible mechanisms Weibo used to carry out censorship.

To collect the deleted (censored) Weibo posts, Tao focused on a set of sensitive users (several thousands) and crawled their timeline every minute over a two-month period in 2012. Tao found that Weibo was surprisingly fast in identifying and deleting sensitive posts. Most deletion happened within the first hour after the content was posted on Weibo. Tao said they tried to reverse-engineer the possible mechanisms Weibo used to achieve fast censorship. According to Tao, Weibo seemed to be using a keyword-based filter, combined with dedicated human censors. Also Weibo paid closer attention to users who frequently posted sensitive content.

One attendee pointed out that Weibo could potentially pollute Tao's data by intentionally returning incorrect timeline data. Tao said at the time of their study, they ran some validation tests by comparing the content returned from the API and the Web site, and did not find any inconsistencies. Another person asked how they knew the deleted posts were caused by censorship, not other reasons like spam or even self-deletion. Tao said the error message for self-deleted posts and Weibo-deleted posts were different. Also those users they monitored were carefully selected to make sure they were involved in censored discussion before. Thus their content was unlikely to be spam.

Another questioner asked what people would do after they got censored. Tao said he saw people started to perform some obfuscation on their posts, changing the form of keywords, for example, or using keyword substitutions. Someone asked how Weibo censors knew what topics to censor. Tao said there were multiple possible channels: they might get orders from the government to censor certain topics, or based on those sensitive users' recent posts or external resources like oversea news.

Thursday Keynote Address: The White House's Priorities for Cybersecurity

Andy Ozment, Senior Director for Cybersecurity, White House

Summarized by Theodore Book (theodorebook@gmail.com)

Andy Ozment spoke about the Obama administration's priorities for cybersecurity. He identified five basic priorities: protecting critical infrastructure (see Executive Order on Cybersecurity, below), securing the government, engaging internationally, improving incident response, and shaping the future. He emphasized the recent executive order on cybersecurity and summarized its main points.

Securing the Government: The federal government is a large institution with an unknown number of machines, people, and agencies. Work to secure it is being conducted by establishing standards and holding people accountable. There are a series of cross-agency priority goals: First, implementing trusted Internet connections. Currently, they don't know where they are connected to the Internet. They have found tens of thousands of connections, and are finding more all the time, but want to move to around 50. Secondly, they want to implement two-factor authentication, through the use of a smart card that provides both physical and electronic access. The third goal is continuous monitoring. Here, they seek to measure how secure they are—knowing vulnerabilities, and incentivizing higher security.

Engage Internationally: There need to be consequences for those who are trying to intrude, otherwise they will eventually get in. By using the word "intrude" rather than "attack," they are consciously using the language of espionage and not war. This process is extremely slow. They are engaging with the Chinese government, by raising this issue through diplomatic channels and a working group. They are trying to convey that there is a norm of behavior for espionage that distinguishes economic from government espionage. They want to discourage economic espionage by state actors. They are also working with the Russians in a long series of negotiations that have led to a red phone for cybersecurity incidents.

Improve Incident Response: A year ago, they held a national-level exercise on cybersecurity (these have traditionally focused on physical events like earthquakes and hurricanes). They have also been facing a steady year of DoS attacks against the financial services sector. They collected a list of attacked IPs and passed them to ISPs. Originally the process would take two weeks. They can now do it in minutes or hours.

Shape the Future: Attackers have the edge—they can keep trying until they succeed. They want to make things better by focusing on DNSSEC, routing security, building a cybersecurity workforce, and R&D into less vulnerable systems.

Executive Order On Cybersecurity: The recent executive order on cybersecurity has four goals: information sharing, privacy and civil liberties, standards, and the identification of critical infrastructure.

Information Sharing: The administration wants to have parties share information on attacks, so that it becomes possible to understand the scale of a single intruder's activity. They also want to share indications of intrusions, so that if an intruder is caught in one place, he will be caught everywhere. The current goal is for the government to share information with the private sector, not because the government necessarily knows more than the private sector, but because it is easier within current laws. They also want to change government culture and classify less data. The problem is that sharing information can cause that information to lose its value. Even limited releases of information are quickly picked up by adversaries. Still, they are going to share more.

They want to offer an intrusion detection system called Enhanced Cybersecurity Services that uses classified signatures. These signatures are given to private sector enterprises who are certified to store it and who have personnel with security clearances to handle it. A generic infrastructure firm can then run traffic through this black box to block malicious traffic. This is useful for small firms that don't have the in-house capability to analyze malicious traffic.

Privacy and Civil Liberties: Sharing government information with the private sector includes some privacy risks. Recent documents reference the Fair Information Practice Principles, which represent the accepted best practices for these questions.

Standards: Many companies have very poor cybersecurity standards. To improve this, the government is asking companies to share lessons learned from NIST. The goal is to build a framework (not a new set of standards) that collects standards together to provide a comprehensive guide for information security. These standards can become a basis for regulation. For example, regulators of existing industries (such as utilities) will be encouraged to create new regulations to force people to do what the government wants based on these standards.

Identification of Critical Infrastructure: There have already been many efforts (post Sept. 11) to identify critical infrastructure; however, they had more emphasis on physical threats. The goal of the current survey is to identify infrastructure, vulnerable to cyberattacks, whose loss would cause a catastrophic impact. This produces a shorter list that is easier to manage. It also allows government to prioritize companies for regulation and support. They are currently informing companies who made the list.

Legislative Priorities: The Obama administration is looking for more power from Congress to impose its views on cybersecurity. They dislike the idea of allowing states to make their own regulations, and prefer to concentrate power in the executive branch of the federal government so that we won't have 50 different sets of regulations. They want to collect more information from companies, but they also want to ban companies from providing information with personally identifying information, and to restrict the use of information collected by the government in that way.

A number of individuals asked questions after the presentation. Several attendees asked questions relating to the relative value of voluntary or mandatory standards. In reply, Dr. Ozment stated that there is no appetite for mandatory standards. People feel that a top-down approach would be harmful rather than helpful. They are then asking regulators to look at the voluntary standards, which they may then impose through regulation. A past attempt to get Congress to pass a law enabling the administration to impose standards failed. He also indicated that there are some problems with FISMA (the existing standards, which the questioner had criticized). Some organizations do a great job within this framework. Others have devolved into a regulation-compliance approach. Some problems are with the law, others with procedures required by the executive branch. They are trying to update it, and update their internal procedures.

In May 2011, they did not propose specific regulations, but the authority to regulate. The executive order is a good alternative solution. It allows for cooperative development of the framework which can then be imposed on many companies through existing regulators, although there will be some holes. Regarding other companies, three agencies were tasked to produce reports suggesting how they might be forced to act according to the administration's desires. They looked at various incentives and came up with nine. Some were: using the insurance industry by making the standards a possible factor in setting rates; using the rate recovery mechanism—regulated utilities could charge more to cover security expenses; prioritizing government grants and assistance; etc. Some of these will have to wait until the framework is done.

When asked whether the proposal to provide classified signatures to certain providers was potentially anticompetitive, Dr. Ozment replied that this program has been piloted with the defense industrial base. There is no limitation on service providers. Anyone who is willing to meet the required standards and provide staff that can clear the background checks can take part.

On the question of whether incident reports should be publicly available, he indicated that we are in a difficult spot. Most companies do not report intrusions. We have to incentivize reporting. This means that a company should see a positive outcome as a result of reporting (e.g., intruder caught). Also, there should not be a significant downside, and for most companies, releasing the reports would be a downside.

In response to a wide-ranging question, Dr. Ozment stated the following: regarding [FDA] regulations prohibiting updates, some areas have a strong culture of safety and reliability that clashes with the culture of security—they prefer not to update. Regarding funding, no budget has been passed since Obama took office. Nonetheless, he believes that cybersecurity funding has increased—he will check on that. The government doesn't have a good way of tracking what it is spending money on, so there is no way for the administration to know what it is spending on cybersecurity. They are trying to figure that out by putting more regulations on government departments and requiring them to report more information. Regarding the recent unauthorized disclosures and what people are reading in the newspapers, he doesn't know what is going on, and could only read talking points in any case. He does want people to be able to trust the government and share their information with the government.

On the question of metrics, he observed that good metrics in cybersecurity are hard to come by. Right now, there is an obvious problem even without metrics. They will deal with metrics when the big things are tackled. On the question of privacy regarding biometric data, he indicated that society needs to define these issues, not just the government. Government can record that consensus. The commerce department released a "green paper" on privacy, which might be worth looking at. When asked about international engagement with allied and neutral countries, Dr. Ozment replied that the administration is helping other countries to develop norms of behavior as to what is acceptable in cyberspace.

Finally, on the question of education, he stated that they have national cybersecurity information month. Most is focused on universities, some on broad national awareness. They can generally raise awareness—it is more tricky to offer useful advice to individuals.

Large-Scale Systems Security II

Summarized by Frank Imeson (fcimeson@gmail.com)

You Are How You Click: Clickstream Analysis for Sybil Detection

Gang Wang and Tristan Konolige, University of California, Santa Barbara; Christo Wilson, Northeastern University; Xiao Wang, Renren Inc.; Haitao Zheng and Ben Y. Zhao, University of California, Santa Barbara

Gang Wang explained that a Sybil is a fake identity owned by an adversary and is maliciously controlled. Sybils have infiltrated social networks in a big way with 14.3 million on Facebook and 20 million on Twitter. The types of attacks Sybils can execute range from spamming unwanted advertisements, malware, phishing, stealing user information, and even political lobbying efforts have been made to try to release fake headlines about

Obama. One might assume that a Sybil's friends list would consist of mostly other Sybils, but this is not the case and in fact it is often the case that by the time a Sybil requests you as a friend they already have 20 or more of your friends in common with you, which on the surface makes them seem more legit to you and even to a static analysis of the graph.

Wang et al. proposed an alternative to static graph analysis, which is to monitor the time and click events to distinguish between normal and Sybil users. This is motivated by the intuition that a Sybil is goal oriented and time limited so one might expect a pattern and efficiency to a Sybil's clicks. They investigate this approach by building a classifier that takes the clickstream (click time and events) as input and is trained offline with ground truth or trained online with input from a set of trusted users. Results of the classifier trained with ground truth only had 3% false negatives and 1% false positives. They shipped their software to Linkedin and Renren, where Linkedin trained the classifier with a ground-truth set of 40k users' clickstreams and was able to flag 200 new Sybils. Renren used the classifier on 1M users, flagged 22k suspicious users, and identified a new attack (embedded URLs in images). Wang concluded by stating that good Sybil detectors force the Sybils to slow down their click speed, mimic normal users and thus turn a beast into a puppy.

Siddharth Garg, University of Waterloo, asked how they cluster the graph? Wang answered that they use automatic clustering and just need to choose the resolution—too small and there's a loss of generality, too large and they lose accuracy. Garg asked how this software is effective over different data sets. Wouldn't it have to be unsupervised since there is no ground truth? Wang said that for this case we would need to generate a small data set to use for ground truth. Simon Chung (Georgia Tech) said that if the Sybil must limit its click speed, can it achieve the same throughput with many parallel Sybils? Wang answered that this is possible and is also why they do not simply classify Sybils by the time intervals between clicks, but also look at event transitions.

Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness

Devdatta Akhawe, University of California, Berkeley; Adrienne Porter Felt, Google, Inc.

Devdatta Akhawe began by explaining that this study was conducted on data collected from Google's Chrome and Mozilla's Firefox from users who have opted in to sharing "Telemetry" data. The information about how the user responds to the warning is recorded in the browser and shared with Google or Mozilla. The study "Bridging the Gap in Computer Security Warnings," Bravo-Lillo 2011, states that "most people don't read computer warnings, don't understand them, or simply don't heed them." Because this was contradictory to Akhawe et al. findings, they conjectured that the original studies got these results because they were conducted in a lab environment, used trusted computers, presented the user with text-only warnings, and only required one click confirmation. Today's warnings are more engaging, including pictures, offering lay content with a link to read more details, and often requiring a multi-step override such as asking: are you really really sure?

Click-through rate is the ratio of warnings ignored over warnings shown, and they claim that an ideal click-through rate is 0% (all warnings should be heeded). This ideal rate should motivate content providers to fix their Web content in the case of false positives and thus would also alleviate users of annoying false warnings. The results show an interesting difference between Firefox, Chrome, Windows, Mac OS and Linux users. For example, Firefox had a lower click-through rate on both phishing and malware. Differences like this could be due to the amount of effort (number of clicks) it takes the user to ignore the warning but in the case of Firefox it only takes one click to ignore compared to Chrome's two clicks to ignore. Linux users also show a much higher click-through rate than Windows or Mac OS users. Also users of the beta or dev releases of the browsers show higher click-through rates. Which begs the question: "Does a greater degree of technical skill correspond to reduced risk aversion?" Akhawe states that this data shows that users do actually heed warnings, but the design does impact the users' behavior.

Frank Imeson (University of Waterloo) asked if there are times when a warning should be ignored and, if so, wouldn't that make the ideal click through rate non zero? Akhawe said that if there are false positives then the browser should ignore them and/or the content provider should fix their content, but this is a very long argument to be discussed more offline. Someone else commented that improvements are the result of improved warnings and an increase in public education. Is there a way to tease out the effects of education from the results? Akhawe said he doesn't know how they could do that but it would be useful information. Someone else asked whether there was a way to assess false positive rates. Akhawe replied not at the moment.

An Empirical Study of Vulnerability Rewards Programs

Matthew Finifter, Devdatta Akhawe, and David Wagner, University of California, Berkeley

Akhawe stayed on stage to present his work on reward programs for finding bugs. Google and Mozilla both offer a reward-based program to users who sign up to find bugs for their browser software. This study analyzes the difference between the traditional approach of hiring an engineer to find bugs compared to outsourcing this task to willing and able end users. If the user is able to find a bug, he or she is rewarded. This reward may be proportional to the severity of the bug as with Google; sometimes Google also revisits the severity assessment of the bug and, if they think the bug was more important than they originally thought, retroactively award more money to that user.

The finding states that vulnerability reward programs (VRP) are cost effective: Google spends about $485 per day while Mozilla spends about $658 per day, which is comparable to an engineer's salary doing the same job. VRPs have found on average more bugs than internal engineers for both non-critical and critical bugs: Mozilla reports that its VRP found 148 bugs compared to the 48 bugs found by the internal engineers, while Google reports that more critical bugs were found by VRPs than by their internal engineers. Akhawe also showed that Chrome has a smaller proportion of bugs considered critical than Firefox, which he hypothesizes to be because of privilege separation in Chrome. Akhawe concluded that Chrome and its VRP is more popular than Firefox. Google's VRP finds more bugs, has a shorter time to patch than Mozilla's, and has shown good repeat participation by users.

Someone from the University of Maryland complimented Akhawe on his talk and asked how VRP compares with black market reward programs. Akhawe responded that although black market rewards are higher, the required commitment is greater since they are looking for a working exploit. He added that people are generally good and black markets do attract the majority of bug finders. Jason Jones (Airborne Networks) asked about the effect of having programs like ZDI buying up exploits for Chrome and Firefox with respect to this work. Akhawe said that he doesn't know enough about ZDI but it would be interesting to take a look at. Someone else asked whether he had any data on false positives and had the time wasted on these cases been factored into the cost-effectiveness of VRP. Akhawe replied that he didn't have any data on false positives but in his conversations with Mozilla and Google no one had ever mentioned false positives as an issue. Jerry Tyson (Facebook) asked how this could work for Web apps and what the differences would be. Akhawe said that he has thought about it, thinks there would be advantages and disadvantages, and would love to get his hands on data from Facebook. Tim Fraser (DARPA) said that assigning metrics for measuring security is hard, but would the amount of money spent on the black market for these bugs be a good metric? Akhawe replied that black market money might be indicative but that metrics are difficult; the lack of spending by a vendor on bugs, however, may indicate a lack of security.

Applied Crypto II

Summarized by John Scire (jscire@stevens.edu)

Secure Outsourced Garbled Circuit Evaluation for Mobile Devices

Henry Carter, Georgia Institute of Technology; Benjamin Mood, University of Oregon; Patrick Traynor, Georgia Institute of Technology; Kevin Butler, University of Oregon

Henry began his presentation by discussing the current abilities of smartphones to perform SMC, or secure-multiparty computation. SMC involves two or more parties trying to securely evaluate some function without revealing their inputs. Smartphones now are limited in several aspects, one of which is computational power, which SMC heavily requires. This is mainly due to the large amount of computation and memory necessary for garbled circuits, which are circuits constructed to perform the evaluation of an SMC function and whose inputs at each gate is obfuscated in some way. To solve this problem, Henry and his team devised a protocol that would push most of this heavy computation to the cloud, specifically in the two-party scenario, in a way that also allows all parties to be assured of the correctness and validity of the output.

The protocol uses Kreuter et al.'s maliciously secure SMC technique along with consistency checks and an outsourced oblivious transfer mechanism. To further describe the protocol, Henry provided the following scenario (featuring Alice, a Web server, Bob, and the cloud): (1) the construction of circuits by Bob, (2) an outsourced oblivious transfer involving all three parties to generate key information as well as Alice generating her garbled input, (3) the generation of Bob's input, (4) the evaluation of circuits by the cloud, and finally (5) the delivery of output. Henry mentioned that these steps retain all of the security checks used in Kreuter et al.'s previous work, but the formal proofs of security for the whole protocol are in their technical report, which is cited in the paper.

To test this protocol, Henry and his team put Kreuter et al.'s work onto servers and had a Galaxy Nexus phone connected to these servers. They then created a bunch of test mobile applications that use classic SMC functions, such as the Millionaires' Problem and edit distance, and ran these applications with and without the help of the servers. As a result, they saw that smaller inputs actually ran better on the device by itself, but of course larger inputs were dramatically slower on just the mobile device. The addition of the cloud performing the computation introduced a 98.9% speedup in terms of total execution time over just using the mobile device in the edit distance application with an input size of 128.

Someone asked whether anything would actually be problematic with Alice colluding with the cloud. Henry responded that allowing Alice and the cloud to collude could break some of the consistency checks that are in the protocol, which would cause Bob to lose assurance of the protocol. He also said that this is something that they could work on to improve.

On the Security of RC4 in TLS

Nadhem AlFardan, Royal Holloway, University of London; Daniel J. Bernstein, University of Illinois at Chicago and Technische Universiteit Eindhoven; Kenneth G. Paterson, Bertram Poettering, and Jacob C.N. Schuldt, Royal Holloway, University of London

Jacob first presented a brief introduction to TLS, which is used widely today for secure HTTP connections, and the RC4 stream cipher. Transport Layer Security, or TLS, consists of two protocols: the Handshake protocol and the Record protocol. The

Handshake protocol is used to establish the connection, whereas the Record protocol deals with the encryption of the payload of the packet. This research, however, only deals with the Record protocol because this is where RC4 is used. RC4 involves two algorithms: key scheduling and key generation. Key scheduling initializes a byte permutation using a key. Key generation then further permutes this byte permutation to create the keystream used for encryption. As Jacob mentioned, RC4 is used in more than 50% of all HTTPS connections, despite known statistical weaknesses. Using these known weaknesses, Jacob and his team created two plaintext-recovery attacks against RC4.

The first attack Jacob and his team made uses single-byte biases that exist in the first 256 bytes of the RC4 keystream. To do this, they first created a keystream byte distribution using many 128-bit RC4 initial keys. They then took these keys byte by byte and XORed them with a chosen plaintext candidate byte in order to get an induced distribution. From there, they just computed the most likely plaintext byte for each of the byte positions. Jacob mentioned, however, that this attack required the same plaintext to be encrypted under different keys each time. Jacob and his team found several ways to make this happen, such as by causing a client to continuously request access to a secure Web site via a session renegotiation or resumption. The second attack used a similar approach but involved known biases that exist within consecutive bytes in the entire RC4 keystream. Jacob pointed out that the full details of how this worked were in the paper. Other than the difference in the algorithm for the attack itself, this second attack requires the same plaintext to be encrypted with the same RC4 keystream. This precludes the need for any type of session renegotiation such as was required in the first attack. This attack is also not restricted to the first 220 bytes of the plaintext.

In terms of performance, the first attack showed an increase in percentage of plaintext recovered with an increase in the number of sessions used. In fact, Jacob and his team were able to achieve a plaintext recovery rate of 100% with a very large number of sessions. As for the second attack, the recovery rates were high and scaled with the increased number of same plaintext copies. Despite these high recovery rates, both attacks required a vast amount of traffic to succeed and so were not practical; however, Jacob still suggested stopping the use of RC4 altogether as the most efficient way of preventing all of these attacks.

Someone asked whether these problems were caused by the TLS implementation or by TLS's interaction with RC4. Jacob said these problems were in fact due to how RC4 was implemented. The same person asked whether RC4 should still be used to protect credit card transactions online, as using RC4 is part of the standard for dealing with credit card information. Jacob said it depends. If you were using TLS 1.0 unpatched against a BEAST attack, for example, he would recommend just using RC4.

PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation

Ben Kreuter, University of Virginia; Benjamin Mood, University of Oregon; Abhi Shelat, University of Virginia; Kevin Butler, University of Oregon

Ben first gave an overview of previous work on secure two-party computation. He pointed out that previous solutions to creating toolsets for two-party secure computation worked, but they suffered in their scalability. To fix this, Ben and his team developed not only a method to scale these secure computations, but also an entire library to do this called PCF.

Ben then went into several optimizations of previous work that make up PCF. One such example is that of reducing the storage size of circuits, particularly the storage of wire values, during runtime. Originally, a high-level language would be used to write the protocol and then compiled into a circuit; however, circuits can grow immensely during runtime depending on the protocol, such as with wire values. During runtime of a circuit, a table would be created for every wire, and then values would be put into the table entries. This creates a growing memory requirement that scales with the worst case to running time. Ben and his team used a simpler approach that overwrites wires when they are not needed using high-level information that the compiler can provide. Another improvement Ben discussed was that of PCF's flexibility with other languages. PCF can actually support any language for two-party computation. A developer, for example, could simply use standard C to program a protocol without adding any additional changes to the C language. As Ben put it, PCF can be thought of as simply writing and running a normal program.

Using this new tool, Ben and his team were able to handle billions of gates for a circuit. They were also able to reduce circuit file sizes and compile times by large orders of magnitude. Interestingly enough, Ben said that the actual bottleneck was in running the protocol itself.

Someone asked whether they ran into any counterexamples regarding the assumptions they made about the way that they were doing loops via backwards branches. Ben replied that they have not yet found any counterexamples, but they do have a backup plan if need be and a way to carry out the plan. Another person asked how they avoided information leakage if they are not evaluating the full depth of the circuit. Ben responded that only the branches in the forward direction can depend on private inputs. He added that for loops they rely on the user's ability to end the loop and thus do not terminate the loop if it happens to run infinitely, just like running a program.

Protecting and Understanding Binaries

Summarized by Xinyang Ge (xxg113@cse.psu.edu.ge)

Control Flow Integrity for COTS Binaries

Mingwei Zhang and R. Sekar, Stony Brook University

Awarded Best Paper!

Mingwei noted that control flow integrity (CFI) can mitigate attacks like buffer overflow attacks and return-oriented programming (ROP) that need to subvert the original control flow; however, previous CFI implementations rely on the compiler's help or debug information. Mingwei said that their work can apply CFI enforcement on stripped binaries.

The first challenge was to disassemble the binary. On architectures like x86, the instruction length is varied; there are "null" gaps between code, which might be interpreted as instructions during disassembling. The authors combined linear disassembling with recursive disassembling to correctly identify gaps among code sections.

Binary instrumentation also requires transparency to existing code and maintaining the correctness of the original execution. To enforce control flow integrity over executable as well as all dynamically loaded libraries, they instrumented the Global Translation Table (GTT) that is used to map an indirect target with routing address in a different module. To keep the GTT updated, they modified the loader by adding 300 SLOC.

To evaluate the effectiveness of CFI enforcement, they proposed a metric called average indirect target reduction (AIR) that quantifies the fraction of eliminated indirect targets. They compared their techniques with others and showed the effectiveness of eliminating unnecessary indirect targets. To test the correctness of implementation, they applied their approach over more than 300 MB of binaries and the result was that none of them was broken during binary rewriting. Certain optimizations like branch prediction and adding address translation have been applied to the original implementation to reduce the overhead.

Ian Goldberg asked about how the gap is accurately identified. Mingwei answered they do not accurately identify the gap and it is possible the disassembler might mistakenly disassemble the gap. Because the gap would not be executed, it should be fine. Eric Bodden asked about self-loading libraries. Mingwei answered that all of the libraries should be translated in advance or CFI could not be enforced. And they haven't taken care of a self-loaded library so far.

Native x86 Decompilation Using Semantics—Preserving Structural Analysis and Iterative Control-Flow Structuring

Edward J. Schwartz, Carnegie Mellon University; JongHyup Lee, Korea National University of Transportation; Maverick Woo and David Brumley, Carnegie Mellon University

Edward first asked a question about whether researchers would like to read assembly or high-level language code like C. The answer is obvious: C code is much easier to understand than assembly code, and there are many existing techniques that require source code to do static analysis. Thus, their work focused on recovering the high-level abstractions from machine code.

The authors proposed two desired properties of decompilation: effective abstraction and correctness. To illustrate abstraction effectiveness, Edward showed two code examples doing the same thing, one using "goto" and the other using "while". To realize effective abstraction, they divided the decompiler, named Phoenix, into several components and recovered the control flow of the original program. A diagram illustrated how the decompiler works: (1) CFG recovery, (2) type recovery, (3) control flow structure, and (4) source code output. They captured the types by extracting the semantics of instructions. For instance, "movl (%eax), %ebx" reveals %eax is a pointer to type A while %ebx is of type A. With types, they further recover the control flow and generate source code. In order to preserve structuredness of source code, they apply iterative control flow structuring for source code generation. The aim is to minimize the use of "goto".

For evaluation, they showed an example decompilation of a short program and demonstrated the effective abstraction their decompiler can achieve. Then they launched some large-scale experiments with other decompilers (e.g., Hex-Rays, Boomerang) on GNU coreutils. They use two metrics to measure Phoenix: correctness and structuredness. The result turned out 50% of tested programs can be correctly executed and less goto's are used compared to other decompilers (details can be found in their paper).

Someone from UC Berkeley asked about whether their work focused on languages other than C. Edward answered currently their work focuses on C. Someone else asked about obfuscation or handwritten assembly. Edward said they are only looking at assembly directly from a compiler. Michael from UC Berkeley believed compiler optimization could change control flow. Edward said it is possible but if it represented the same logic, things should be fine. Scott Karlin (Princeton) suggested a further use case of detecting source code plagiarism. Finally, a researcher from Cisco asked whether they have tried multiple phases of compiling and decompiling using their tools. Unfortunately, the answer was no.

Strato: A Retargetable Framework for Low-Level Inlined-Reference Monitors

Bin Zeng and Gang Tan, Lehigh University; Úlfar Erlingsson, Google Inc.

Normally, attacks are launched by triggering existing bugs inside programs using user input. Previous countermeasures include data execution protection, address space layout randomization (e.g., PaX), and inlined reference monitors (IRM). An IRM is nothing but placing security checks inside programs. Most IRMs are implemented at a low level, which is difficult to reuse. Also, low-level instrumentations are restricted to a certain architecture and difficult to port. Thus, their work performs IRM rewriting at the Intermediate Representation (IR) level.

The challenge of doing IRM rewriting at IR is risky because the compiler is not reliable. For instance, the compiler might optimize the security checks out at the backend. So they intentionally add checks that are respected by the compiler and also verify that these checks are preserved after compilation is done. To illustrate how security checks are added to IR, Bin gave an example of IR code with checks added. Additionally, they also did optimizations on the security checks including removing redundant checks.

To evaluate, they measured the performance on SPEC2k and portability by using same security checks on both x86 and x86-64. The average performance overhead was about 21%. For portability, the same instrumentation could work on both x86 and x86-64.

Someone asked about whether the security check is really ISA independent. Bin answered it actually depends on what the security check is. In fact, IR itself is not ISA independent. Ben Livshits (Microsoft Research) asked why the performance overhead is that high. Bin said intuitively this is related to the number of security checks placed, but they haven't measured what really incurs the overhead.

Invited Talk

Confessions of a "Recovering" Data Broker: Responsible Innovation in the Age of Big Data, Big Brother, and the Coming Skynet Terminators

Jim Adler, VP of Products, Metanautix

Jim Adler began his invited talk by introducing his company Metanautix. Metanautix is working on building a next generation big data management and analysis system. It has already built massive data analysis systems for many large enterprises such as Google and Facebook.

Based on his experience, Jim introduced the data supply chain. The huge amount of data from government, commercial, and self-reporting can generate huge value and are powerful for applications in transportation, marketing, etc.; however, only few data collectors are regulated. Those unregulated uses can be easily abused by powerful people, and the hugeness and variety of data makes the world have less anonymity.

Through comparing EU rights and US torts, Jim asked, how do we unpack privacy and distinguish private from public? He further used place, player, and perils (3P) to characterize privacy issues. To describe the relationships among 3P, he concluded that player power gaps are proportional to secrecy and have an inverse relationship to trust. He further gave us an example of how his Felon predictor works (http://bloom.bg/1eMtnug) determining whether a person had committed a felony using other information in the database. He showed that the classifiers depend on policy as much as technology. Finally, he concluded that now government doesn't trust people but does trust machines.

A few attendees asked whether it is illegal to share private information on the market. Jim said it depends on what is privacy and what is public. Supermarkets usually do not share their customers' information with others. Other attendees were also curious about how to know which info is correct among huge data. Jim said through the data chain and huge data correlation, we have some mechanisms through which we can infer the valuable data. The world is shrinking in the information era, and we need to respect the data. Some people were worried about their privacy and asked whether we have choice to protect our privacy. Jim said that we need new policy now to deal with privacy protection. And we need better behaviors to protect our own privacy.

Current and Future Systems Security

Summarized by Sven Bugiel (bugiel@cs.uni-saarland.de)

On the Security of Picture Gesture Authentication

Ziming Zhao and Gail-Joon Ahn, Arizona State University and GFS Technology, Inc.; Jeong-Jin Seo, Arizona State University; Hongxin Hu, Delaware State University

Ziming Zhao presented his research on the security of picture gesture authentication (PGA) as deployed, for example, in the latest version of Microsoft's Windows 8 operating system. In PGA, users choose a background picture (from local storage) and perform gestures on this picture, such as tapping, drawing a circle, or drawing a line. The order, precision, and direction of those gestures then form the user password for authentication. To better understand the security of this new authentication mechanism, Ziming and his co-authors were first interested in better understanding the user-choice for background pictures and gestures. Using the results of this investigation, they devised and evaluated an automated attack framework to successfully break users' gesture passwords.

To investigate the users' choice of passwords (i.e., pictures and gestures), the authors conducted a user-study with two user-groups. The first group consisted of 56 computer science undergraduate students from Arizona State University, uniformly male, which used PGA for accessing class materials on the university Web site. The second group consisted of 762 participants recruited over public channels such as crowdsourcing, and their task was to emulate logging in to their online banking Web site using PGA. The study yielded that, from all picture categories, pictures depicting people are most commonly chosen since they are easier to remember, and that there is a strong relationship between the user's personality and his choice for his background picture. More importantly, the study showed that gestures are generally drawn around distinct points of interest, such as objects, shapes, or preeminent colors, and that the patterns for drawing gestures are very similar among different users.

Ziming and his co-authors applied these insights to design and implement an automated attack framework to break users' passwords. At the heart of their framework is a location-dependent gesture selection function that models and simulates the users' selection of gestures around and between points of interest. Evaluation of the attack framework based on the passwords collected from the user-study showed that the authors could successfully break between 24% (group 2) and 48% (group 1) of the passwords. This difference in success rate is explained by the lower security-sensitive context for the first group (access to class material), which resulted in simpler gestures (e.g., three times tapping a point of interest). Moreover, the evaluation showed that the attack success rate is noticeably higher for simple pictures with few points of interest and for portrait pictures with more predictable gestures. When tested as real-life online attacks on Windows 8 (i.e., only five attempts on guessing the gesture password) for passwords of the second group, the authors were still able to break 2.6% of the passwords.

The data sets of the user-study are available online at http://sefcom.asu.edu/pga/, and an example tool for measuring the gesture password strength is provided at https://honeyproject1.fulton.asu.edu/stmidx.

Chris Thompson (UC Berkeley) asked about the recall over time of gesture passwords and, further, if the two user groups are not too biased and participants of the first group are incentivized to choose weaker passwords. Ziming replied that they evaluated memorability of gesture password for the first group and the results are presented in the paper. The two groups were chosen on purpose in this configuration, and although users of group one did change their passwords to weaker ones, why they did so is unclear. Some feedback indicated that the weaker passwords were easier to use on smartphones. David Wagner asked whether the authors compared their real-life success rate of approximately 3% to the best attacks on text passwords. Ziming explained that they have not yet compared their results, but that the password space for picture gesture authentication is bigger than for text passwords, and this space could be further increased by allowing more gestures.

Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization

Rui Wang, Microsoft Research Redmond; Yuchen Zhou, University of Virginia; Shuo Chen and Shaz Qadeer, Microsoft Research Redmond; David Evans, University of Virginia; Yuri Gurevich, Microsoft Research Redmond

Yuchen Zhou presented his results in uncovering implicit assumptions by authors of authentication services' SDKs that can potentially compromise the security of applications that use those SDKs. As a result of this research, Yuchen and his co-authors were able to discover flaws in Facebook's authentication service and in the OAuth 2.0 specification.

Applications, today, are increasingly empowered by online services. One very prominent example is single sign-on (SSO) services offered by Facebook or Windows Live. To incorporate those services into their applications, developers are provided with SDKs and corresponding documentation on how to use the SDKs. Yuchen and his co-authors posed the question, whether the application is secure if the developer adheres to the SDK's documentation. He illustrated that this is not the case, by showing a demo video of an attack in which a malicious app is able to steal credentials retrieved from the Windows Live SSO service and use those credentials to impersonate itself as the legitimate user. Yuchen showed that such security issues can be traced back to implicit assumptions by the SDK developers, such as assumptions that are essential for the application's security properties and are not clearly stated in the SDK documentation, or that relate to how the SDK should be used.

To systematically discover such implicit assumptions in SDKs and their associated documentation, the authors of this paper built semantic models that capture both the logic of the SDK and the essential aspects of underlying runtime systems. To be able to consider all possible apps that can be built with an SDK, these models consider both the client and the service side. The semantic models, together with explicitly captured assumptions and security assertions (i.e., desired properties such as authentication or authorization), form the input to a BOOGIE-based verifier. In an iterative process in which the model is refined or new assumptions are added, the final assumptions for this model are derived.

Applying this approach to explicate the three concrete examples of Facebook SSO PHP SDK, Windows 8 SDK for modern apps, and Windows Live connect SDK, Yuchen and his co-authors were able to uncover implicit assumptions that lead to a change of the Facebook SDK, a revision of the Windows Live SDK documentation, and an addendum to the OAuth 2.0 standard. Moreover, the authors conclude that due to these implicit assumptions, a majority of the tested apps—for example, Facebook's showcase apps—were vulnerable to attacks, and Yuchen illustrated this with concrete vulnerabilities for the Facebook SDK and Windows Live SDK.

Felix Lindner (Recurity Labs) asked about the efficiency of this approach versus a good Web-application pen tester. Yuchen replied this is a guided approach to better understand the system and find vulnerabilities. Penetration testing, on the other hand, is rather a black box testing to find vulnerabilities. Yuchen argued that their approach is more systematical but might help increase the efficiency of penetration testing. Someone asked whether the authors considered applying their approach more generally instead of only to SSO SDKs. Yuchen answered that their approach can definitively be generalized and applied to other models like payment, but they focused for now on SSO. Adrienne Porter Felt (Google) followed up on the different responses Yuchen received from the SDK providers and wondered whether updating an SDK documentation to include implicit assumptions is really enough. Yuchen replied that changing the SDK is definitively the best solution, because it is unclear whether developers adhere to the SDK documentation. But that is not always possible and thus documentation updates should be more strikingly propagated to developers to update their code.

Enabling Fine-Grained Permissions for Augmented Reality Applications with Recognizers

Suman Jana, The University of Texas at Austin; David Molnar and Alexander Moshchuk, Microsoft; Alan Dunn, The University of Texas at Austin; Benjamin Livshits, Helen J. Wang, and Eyal Ofek, Microsoft Research

Suman Jana presented a solution for a more fine-grained access control model for augmented reality (AR) applications, which simultaneously allows for a higher scalability of these applications. Suman first illustrated, based on different, popular examples, such as the SoundWalk app or Google Glass, how AR applications abstractly operate: AR apps retrieve raw input from sensors such as the video camera, then apply object recognition algorithms (e.g., to detect hand gestures), and finally render the raw input augmented with virtual objects back to the screen. Currently, AR apps implement this pipeline by themselves and do not rely on operating system support.

Suman explained that this current status has two important drawbacks: first, because the applications retrieve raw, rich input, there is a high privacy risk. He illustrated this based on a face-recognition app that receives raw video camera streams and thus can also scan the background to discover, as an example, whiteboards full of confidential information. Second, the current AR application model does not allow two AR apps to run concurrently on the same hardware and hence does not scale.

The solution Suman presented is based on operating system support for augmented reality in the form of so-called “recognizers.” A recognizer recognizes real-world objects from raw inputs (e.g., face or gesture recognition). AR applications can subscribe to recognizers and retrieve a stream of preprocessed data (e.g., the hand gestures performed or the recognized faces). Because applications do not retrieve raw input streams anymore, this enables a least-privilege access control for AR applications. To explain to the user which data an AR application receives, Suman and his co-authors introduced “privacy goggles,” which previews to the user the filtered output; Suman provided different examples of privacy goggles in his presentation. Moreover, since the preprocessing of the recognizer can be offloaded and its output shared between different client apps, this allows for higher scalability of AR apps.

In their evaluation based on 87 Xbox applications, Suman and his co-authors discovered that 94% of the AR apps required access to the skeleton recognizer, used for tracking movements of a human body, and that only four recognizers (skeleton, person texture, voice command, and hand position) together cover about 90% of the tested applications. Additionally, ten surveys with 50 participants each showed that 86% of the participants considered the recognizer output less privacy-sensitive. Suman presented that even with six apps sharing recognizers, more than 25 fps can be achieved for each app and he additionally showed the offloading of a heavyweight 3D modeling recognizer to an external graphic card. In future work, the authors want to investigate how to securely share the other steps of the processing pipeline among apps (e.g., rendering augmented output to screen) and how to securely support third-party recognizers.

Devdatta Akhawe (UC Berkeley) asked whether moving object recognition to the operating system level would result in a slower application development, because apps might require recognizers not yet available in the operating system and operating systems have slower update cycles. Devdatta wondered how many recognizers would be required for Xbox Kinect apps today, which were not available when the Xbox started shipping. Suman replied that they have no such statistics, but their evaluation shows that the bulk of the apps require only a few recognizers and that corner cases might be addressed in the future with a secure integration of third-party recognizers. Felix Lindner (Recurity Labs) wondered about the 14% of survey participants who were not able to use the privacy goggles despite the clearly unambiguous goggle preview. Suman mentioned that these users were rather boggled by the whole use-case and were unfamiliar with AR. Adrienne Porter Felt (Google) asked about barcode scanners as recognizers. Suman mentioned that this would be easily implementable and in fact they showed how to run a bar code scanner in a privacy-preserving manner in their S&P ’13 paper, “A Scanner Darkly: Protecting User Privacy from Perceptual Applications.”

Hardware and Embedded Security I

Summarized by Bhushan Jain (bpjain@cs.stonybrook.edu)

CacheAudit: A Tool for the Static Analysis of Cache Side Channels

Goran Doychev, IMDEA Software Institute; Dominik Feld, Saarland University; Boris Köpf and Laurent Mauborgne, IMDEA Software Institute; Jan Reineke, Saarland University

Boris Köpf started by discussing how caches improve performance by reducing memory accesses but also jeopardize security by leaking information about the latency for memory lookups. This leaked information can be used to recover secret keys from AES, DES, RSA, and ElGamal. He introduced the three types of cache attacks: timing based, where the attacker can determine the number of cache hits and misses from observing execution time; trace based, where the attacker can see the trace of cache hits and misses by monitoring power consumption; and access based, where the attacker shares a cache with the victim and can find information about the memory locations accessed by the victim. Although some defenses against cache attacks are implemented in hardware, most of them are designed based on the interaction between the hardware and the software. These solutions depend on the cache specifics and the binary executing for security guarantee. CacheAudit helps such solutions to reason about the security guarantee using automatic static analysis of cache-side channels. It derives formal quantitative bounds on the information leaked to the attacker.

Boris then explained the theoretical foundations of CacheAudit. The goal is to compute a bound on the number of possible side-channel observations to give a quantitative security guarantee using program analysis. A binary program is represented by a state transition system where the cache is a part of the program semantics. The problem of computing the set of reachable states is not feasible. Abstract interpretation is a static analysis method where this set of reachable states is soundly over-approximated by using a set of abstract states that are mapped to actual states using a concretization function such that the abstract transition function always delivers a superset of the concrete transition function. Thus the size of superset of set of reachable states represents a bound on the number of reachable states at the end of program termination.

CacheAudit contains different abstract domains representing the states in stack, memory, flags, actual values, and cache hit or miss. It parses x86 code and generates a control flow graph that is traversed by the iterator to access all the possible states that can be reached. Boris did not go into much detail about cache abstract domain due to time constraints. The basic goal of cache abstract domain is to statically predict cache hits and misses. They analyzed the AES-128 implementation from the PolarSSL library. CacheAudit provides different bounds for different attacker models. Few bits are leaked to a timing-based attacker and many bits are leaked to the trace based attacker. If the AES tables are preloaded, the bounds drop to 0 at the point where the table can be entirely in the cache. He directed the audience to the paper for many more results. The source code is publicly available.

Eric asked how to use the CacheAudit reports to distinguish between false positives and actual leakage. CacheAudit helps the security developer prove that the system is secure and allows him to make stronger security claims than before. Monitor the CacheAudit execution and analyze the location where the number of reachable cache states increases above one. Ben Livshits (Microsoft Research) asked about the loss in expressiveness if we go for zero leakage. Boris was not clear on the question. The chair suggested taking the discussion offline as the question and answer apparently needed some discussion.

The complete USENIX Security '13 report, as well as summaries from CSET '13, HealthTech '13, HotSec '13, LEET '13, and WOOT '13, are available online at www.usenix.org/publications/login.

vmware®

EXPANDING THE POWER OF VIRTUALIZATION

From the data center to the cloud to mobile devices.

VMware, the industry-leading virtualization software company, empowers you to innovate by virtualizing infrastructure and streamlining IT operations. With VMware technology, you can efficiently and reliably deliver services that are accessible on demand from any device, anytime, anywhere—making your organization more agile, more responsive, and more profitable.

VMware Academic Program: Committed to strengthening VMware's relationship with the academic and research communities. To learn more, visit **labs.vmware.com**.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

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

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

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

SAVE THE DATE!

FEB. 17–20, 2014 • SANTA CLARA, CA

12th USENIX Conference on File and Storage Technologies

FAST ’14 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.

Full program information and registration will be available soon.
www.usenix.org/fast14