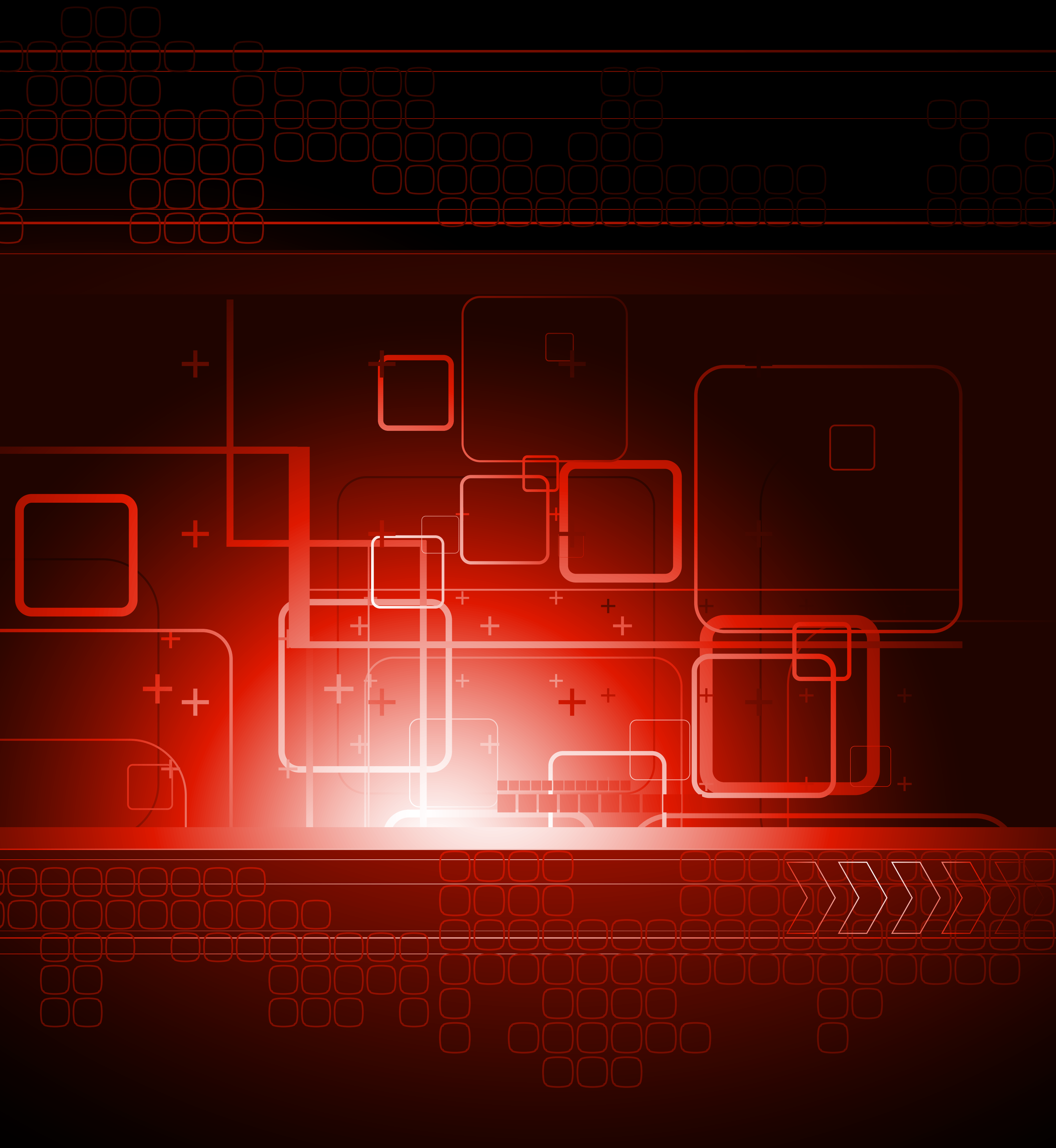
;login: *logout*

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EXCLUSIVE ELECTRONIC EDITION

JANUARY 2014



;login: logout

EXCLUSIVE ELECTRONIC EDITION JANUARY 2014

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Rikki Endsley
rikki@usenix.org

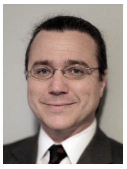
PRODUCTION
Arnold Gatilao
Casey Henderson
Michele Nelson

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

/var/log/manager

Message Not Received

ANDREW SEELY



Andy Seely is the manager of an IT engineering division, customer-site Chief Engineer, and a computer science instructor for the University of Maryland University College. His wife Heather is his init process and his sons Marek's and Ivo are always on the run queue. andy@yankeetown.com.

Communicating about technology can be a greater challenge than the actual technology itself, especially when people from different academic and professional backgrounds need to find common understanding. We sometimes miss that when communication between highly technical people fails, the systems they support can pay the price with poor performance and loss of availability. Understanding communication breakdowns between team members and taking action to ensure that systems are effectively maintained, operated, and repaired is the manager's job. When the manager fails to act to repair fractured communications between team members, the path to a brittle and failure-prone production environment is short.

One of my employees, a BMC Remedy developer on the engineering team, went to the change manager requesting an emergency reboot of the middle tier server during peak hours because something "wasn't right." We knew something wasn't right; performance was off and there appeared to be some sort of intermittent freeze-up of a subsystem on the server that was causing users to get kicked out every few minutes. The change manager asked a reasonable question: How do you know that a reboot will fix it? The developer was sure a reboot would work, but her explanation started out talking about the subsystem and ended up with "it's broke anyway" and "you wouldn't understand." The change manager sidestepped and sent her to the operations manager to get higher approval. The operations manager heard the explanation and interpreted it as a smoke screen for the fact that the developer didn't have any idea what the problem was and was only guessing.

Guessing. In production. At peak. Emergency reboot. Denied.

Communicating Means Really Listening

The developer came to me, frustrated and flustered. "Why don't they understand?" By this time, at least half a dozen people in the technical and management operations teams had different versions of the story, and all were able to agree on one point: Although there was some sort of problem, the application owner didn't understand it.

The developer and I talked for a bit about how no one understands and no one trusts and how the organization was "broken." After she calmed down somewhat, I asked her to describe the current problem and what she felt was the cause. I'm a senior manager in our organization, but I'm also a long-time system administrator with a strong computer science background, and I understand how systems interact and how computer architectures work. After asking a few more probing questions—what are the different subsystems in the server, what services do they provide each other and external servers, what type of data is flowing through the system—what the problem likely was became clear. The subsystems used the local loopback network interface to communicate internally, but a process had a stale socket opened on the loopback that was preventing successful inter-process communication (IPC). The only way to clear the stale socket cleanly would be to reboot the operating system.

"That's what I've been saying all along," the developer said.

We went back to my counterpart in operations together and explained the stale socket and IPC problem and how it was causing intermittent failures of the user interface. The change manager asked, "Why didn't you tell us that the first time?" The emergency reboot during production peak was approved, and ten minutes later the service was performing within normal parameters.

Communication as a Function of the Manager

Ask anyone in an organization and they'll tell you how important good communication is. Some organizations recognize the need for good communication, and have lots and lots of meetings, to varying effect. Others have a strong culture of communication, and as a result have fewer meetings. Ensuring that there is ample opportunity for all types of communication—when everyone needs to know something top-down, when everyone needs to come to consensus, and when someone needs to move information from one brain to another—is the manager's job. Sometimes the manager's job is to communicate directly, and other times the manager must facilitate communication between others. The important skill for the manager is to know the difference.

In the case of the misunderstood developer, my job essentially was to interpret a highly technical subject matter expert's explanation and translate it into non-technical manager-speak. By taking the easy approach of telling the developer to come back later with a coherent explanation, or asking her to send a junior developer whom we could "understand better," I would undermine and demoralize an incredibly strong technical team member who just had trouble being understood, and that approach wouldn't help solve the problem faster.

Even though every job posting we write includes words such as "must be an effective communicator," we don't test for that effectively beyond simply conducting an interview. We shouldn't casually dismiss technical excellence because it is obscured by verbal communications that aren't exactly linear. We should stand up to the management challenge of facilitating communication and set conditions for the success of the employee and of the technical mission. This process starts with trust and is propelled by patience, and it takes a lot of the manager's time and focus. Over time, people get better at trusting each other and understanding each other's intent when the words aren't obvious, but building a culture of communication is done one employee at a time. This effort scales linearly. There's no way to do it faster, but improving communication is part of the manager's job.

Key ideas for managers of technical teams:

- Recognize that technical prowess and effective communications don't always go together;
- However, effective communication and mutual respect are like peanut butter and chocolate. Each is good by itself, but together they're excellent.
- Be a facilitator of communication and help overcome roadblocks. Remember that your employees' jobs are to get the work done, and your job is making your employees as effective at that as possible.
- Understand the different types of communication in your organization, be aware of what's working and what's not, and don't "fix it" only by holding more meetings.

Where Did We End Up?

Our team had a good teachable moment and a few laughs, after we were done being frustrated with each other and after the systems were back in production. Thanks to this episode, we now communicate more effectively in this group, and we have a running joke about how few people on our team actually speak "Remedyese," and how certain developers are not allowed to speak in public without an interpreter present. We have a newfound respect for each other's technical abilities. And we had a chance to talk about computer architectures and IPC, topics that I've made sure my charming and patient non-geek wife gets exposed to every chance I get. Our team still has new communications challenges crop up all the time, and we treat each one the same way—with patience, respect for people, and focus on mission. I'm the manager. That's my job.

Get Salted At

SALTCONF

JAN 28 30 2014

SALTSTACK

The Annual SaltStack Global User Conference

Open to:

Community Members

SaltStack Customers

Partners

Developers

SaltConf will include one full day of pre-conference SaltStack training on January 28th and two full days of conference sessions from January 29 - 30. Get Salted through hands-on labs and training, talks by you, your peers, SaltStack engineers and developers, big keynotes, and lots of hacking and networking.

The SaltConf agenda is suitable for systems administrators, cloud builders and architects, enterprise IT operations, IT directors and managers, developers, and site reliability engineers.

Register now at www.saltconf.com

Code Review for System Administrators

ELIZABETH KRUMBACH JOSEPH

Elizabeth Krumbach Joseph is an Automation and Tools Engineer at HP working on the OpenStack Infrastructure team. She is also a member of the Ubuntu Community Council, one of the two governing bodies of the Ubuntu Project, and on the Board of Directors for Partimus.org, a non-profit in the San Francisco Bay area providing Linux-based computers to schools in need. lyz@princessleia.com

Over the past year, OpenStack [1] has seen fast-paced adoption by major vendors and a staggering growth in its developer community from companies around the world. As a result, the relatively new cloud platform has become quite the popular topic throughout the open source community. The infrastructure used to manage the growth of the project over this time has had to handle this load, while still managing to review and test the code going into the project. In this article, I'll look at how code review practices for OpenStack can be applied to system administration.

The infrastructure [2][3] for the OpenStack project is fully open source and managed by a geographically distributed team of systems administrators from multiple companies who are responsible for the installation and maintenance of the tools used by the OpenStack project. This infrastructure includes the full code review and continuous integration system, plus wiki, pastebin, etherpad, chat bots, and other tools used by project members on a day to day basis. A major boon to the ability for all of us to collaborate effectively is by not only imposing review upon code going into OpenStack itself, but also to all the configuration files and code that we deploy in production for the infrastructure.

The code review and continuous integration system used in OpenStack is built with the needs of OpenStack in mind. OpenStack is essentially a big project made up of many smaller projects that operate largely independently within the OpenStack umbrella. Each project has different team leads, core contributors, and reviewers. As such, the system needed extensive integration testing as part of what is tested before code is committed to the git repositories. We're using Gerrit [4] for the front end code review mechanism and Jenkins [5] as our continuous integration server that launches tests. To glue these together, we use a test worker distributor, Gearman [6], to hand things off to our Jenkins servers. We use a couple other tools to manage the queuing of testing and merging of code (Zuul [7]), and to manage the pool of machines used to do tests (Nodepool [8]), both of which are open source and were developed by the infrastructure team.

Systems administration does not typically need the same kinds of tests that fully integrated code within OpenStack does. Instead, sysadmins on my team use this same system, but we have set up a series of checks to run against all changes to our config files and other code that the team checks in, including:

- flake8 (running pep8 and pyflakes) for our Python scripts
- puppet parser validate & puppet-lint against any changes in Puppet
- XML syntax checking on some XML files where the structure is known

These checks are run as soon as we submit a code or configuration change into the code review system, giving the system administrator feedback within a few minutes about whether their code has passed these basic tests. We regularly assess this list, improve it

References

[1] OpenStack: http://www.openstack.org/

[2] OpenStack infrastructure: http://ci.openstack.org/

[3] OpenStack infrastructure code repository: https://git.openstack.org/cgit/openstack-infra

[4] Gerrit: https://code.google.com/p/gerrit/

[5] Jenkins: http://jenkins-ci.org/

[6] Gearman: http://gearman.org/

[7] Zuul: http://ci.openstack.org/zuul/

[8] Nodepool: http://ci.openstack.org/nodepool.html

and add more when we expand the types of code or configuration files we are submitting so that we get as much benefit as possible from automated tests.

Next the changes are reviewed by contributors to the infrastructure team. As an open source project team, we allow anyone to do basic code reviews, but restrict the higher levels of approval to core project members who have a history of being trustworthy and providing a high level of code review expertise.

This human element of the system is perhaps the most valuable part of having a code review process for system administrators. The process provides an opportunity to have multiple eyes on even simple changes before submitting something that could possibly impact hundreds of active developers from dozens of different companies. A single system administrator is no longer responsible for applying a change. Instead, a team collaboratively reviews and approves it. This collaboration can make for a lower stress, higher reward work environment.

As a distributed team, our review process gives us a great platform for checking in a "Work in Progress" change that we can actively collaborate on by commenting on the changes inside the code review system. Also, team members have an opportunity to see the solution that one of us came up with, and make suggestions for tackling the issue in an entirely different way now that we've seen how the proposed suggestion may work. By having the proposed changes in front of us, we're also able to test code independently of the submitter before giving our approval, which often catches edge cases that are found in our varied personal test environments.

Because all of our changes go through code review, including those from core contributors, there is little technical difference between a submission provided by a new contributor or someone who has been with the project for a long time. Every review is handled independently, and we have the same social requirements for accepting a change (two core reviewers should give their approval). This also means that no one has the ability to commit directly to the code repository. Changes by core members, like those from anyone else, must pass syntax checking and get reviews.

Once the code lands in our git repository, the Puppet master picks it up and it is deployed automatically to the appropriate servers. If the change is to code being run on a server, we use a Puppet mechanism for handling code repositories that regularly checks for updates in the repository and restarts services on changes as needed.

All of these processes have trained members of the team to be collaborative by default, an important thing for a distributed team or one that tends to have different systems administrators focusing on different projects. This has helped tremendously when the limitations for managing systems completely through code review and code repositories are considered.

The first obvious limitation is bootstrapping this process. To get the process going, you need basic servers set up, and when adding new servers to the infrastructure from our pool of OpenStack-based virtual machines, there are still portions we have not been able to automate completely. Also, there is the handling of passwords, SSL keys, and other sensitive data that cannot be made available generally to every member of the project. These limitations are both handled by having a "root" team that has access to creating new servers and to adding, viewing, and manipulating the sensitive data.

We've also had to handle the inevitable problem that comes up where you simply must log into a server for some reason. Perhaps a MySQL database needs a manual edit, or the Puppet agent running on the server has crashed. We may also need to debug something, so shell access to browse logs and run diagnostic tools is essential. To handle this, root team members have access to all the servers, and then access is granted on a server by server basis to team members with expertise in working on specific applications in the infrastructure. We also run public monitoring of our servers via Cacti and have a Puppet dashboard so basic statistics about system resources and the application of changes via Puppet can be tracked and reviewed by any contributor.

Complicated upgrades or migrations also are difficult to manage through a code review system. In these cases, shell access often is required, but our collaborative culture makes it so that we're always working together on these projects. Typically, resolving these situations starts off in an online team meeting in which we flesh out the migration plan in a collaborative editing tool (such as etherpad), then we schedule the maintenance window when multiple team members will be available. Once we get to the maintenance time, we work together on Internet Relay Chat (IRC) to run through the list of tasks defined in the etherpad and work together if anything goes awry.

There are also the inevitable emergencies that crop up from time to time. In these cases, the root admins do have the ability to log in and shut down the Puppet agent and make changes manually to unblock us. The team has been disciplined so that these incidents are rare, as we'd much rather fix it via a commit, and followed up with as soon as the emergency has passed and more core members are available to assess and permanently solve the problem.

Fortunately these limitations are a small percentage of what we encounter, and our primary contact with our systems on a day to day basis is through the code review system. In addition to public Cacti and Puppet dashboard that any contributor can access, we also are diligent about maintaining our team documentation for how we run and make changes to our various services and make sure all our configurations and scripts are browseable by contributors in our public git repository. This makes it relatively easy for new contributors to join our team and get up to speed with our full infrastructure, or simply make a single change to address a pain point in our infrastructure, from adding a new test to the continuous integration system to adding a favicon to the project status page. Documentation also allows our existing system administrators to focus on their core skills and slowly get up to speed with other portions of the infrastructure by reading documentation, doing reviews, and watching other team members work.

This World of Ours

JAMES MICKENS

James Mickens is a researcher in the Distributed Systems group at Microsoft's Redmond lab. His current research focuses on web applications, with an emphasis on the design of JavaScript frameworks that allow developers to diagnose and fix bugs in widely deployed web applications. James also works on fast, scalable storage systems for datacenters. James received his PhD in computer science from the University of Michigan, and a bachelor's degree in computer science from Georgia Tech. mickens@microsoft.com

Sometimes, when I check my work email, I'll find a message that says "Talk Announcement: Vertex-based Elliptic Cryptography on N-way Bojangle Spaces." I'll look at the abstract for the talk, and it will say something like this: "It is well-known that five-way secret sharing has been illegal since the Protestant Reformation [Luther1517]. However, using recent advances in polynomial-time Bojangle projections, we demonstrate how a set of peers who are frenemies can exchange up to five snide remarks that are robust to Bojangle-chosen plaintext attacks." I feel like these emails start in the middle of a tragic but unlikely-to-be-interesting opera. Why, exactly, have we been thrust into an elliptical world? Who, exactly, is Bojangle, and why do we care about the text that he chooses? If we care about him because he has abducted our families, can I at least exchange messages with those family members, and if so, do those messages have to be snide? Researchers who work on problems like these remind me of my friends who train for triathlons. When I encounter such a friend, I say, "In the normal universe, when are you ever going to be chased by someone into a lake, and then onto a bike, and then onto a road where you can't drive a car, but you *can* run in a wetsuit? Will that ever happen? If so, instead of training for such an event, perhaps a better activity is to discover why a madman is forcing people to swim, then bike, and then run." My friend will generally reply, "Triathlons are good exercise," and I'll say, "That's true, assuming that you've made a series of bad life decisions that result in you being hunted by an amphibious Ronald McDonald." My friend will say, "How do you know that it's Ronald McDonald who's chasing me?", and I'll say "OPEN YOUR EYES WHO ELSE COULD IT BE?", and then my friend will stop talking to me about triathlons, and I will be okay with this outcome.

In general, I think that security researchers have a problem with public relations. Security people are like smarmy teenagers who listen to goth music: they are full of morbid and detailed monologues about the pervasive catastrophes that surround us, but they are much less interested in the practical topic of what people should do before we're inevitably killed by ravens or a shortage of black mascara. It's like, websites are amazing BUT DON'T CLICK ON THAT LINK, and your phone can run all of these amazing apps BUT MANY OF YOUR APPS ARE EVIL, and if you order a Russian bride on Craigslist YOU MAY GET A CONFUSED FILIPINO MAN WHO DOES NOT LIKE BEING SHIPPED IN A BOX. It's not clear what else there is to do with computers besides click on things, run applications, and fill spiritual voids using destitute mail-ordered foreigners. If the security people are correct, then the only provably safe activity is to stare at a horseshoe whose integrity has been verified by a quorum of Rivest, Shamir, and Adleman. Somehow, I am not excited to live in the manner of a Pilgrim who magically has access to 3-choose-2 {Rivest, Shamir, Adleman}, mainly because, if I were a bored Pilgrim who possessed a kidnapping time machine, I would kidnap Samuel L. Jackson or Robocop, not mathematical wizards from the future who would taunt me with their knowledge of prime numbers and how "Breaking Bad" ends.

The only thing that I've ever wanted for Christmas is an automated way to generate strong yet memorable passwords. Unfortunately, large swaths of the security community are fixated on avant garde horrors such as the fact that, during solar eclipses, pacemakers can be remotely controlled with a garage door opener and a Pringles can. It's definitely unfortunate that Pringles cans are the gateway to an obscure set of Sith-like powers that can be used against the 0.002% of the population that has both a pacemaker and bitter enemies in the electronics hobbyist community. However, if someone is motivated enough to kill you by focusing electromagnetic energy through a Pringles can, you probably did something to deserve that. I am not saying that I want you dead, but I am saying that you may have to die so that researchers who study per-photon HMACs for pacemaker transmitters can instead work on making it easier for people to generate good passwords. "But James," you protest, "there are many best practices for choosing passwords!" Yes, I am aware of the "use a vivid image" technique, and if I lived in a sensory deprivation tank and I had never used the Internet, I could easily remember a password phrase like "Gigantic Martian Insect Party." Unfortunately, I *have* used the Internet, and this means that I have seen, heard, and occasionally paid money for every thing that could ever be imagined. I have seen a video called "Gigantic Martian Insect Party," and I have seen another video called "Gigantic Martian Insect Party 2: Don't Tell Mom," and I hated both videos, but this did not stop me from directing the sequel "Gigantic Martian Insect Party Into Darkness." Thus, it is extremely difficult for me to generate a memorable image that can distinguish itself from the seething ocean of absurdities that I store as a result of consuming 31 hours of media in each 24-hour period.

So, coming up with a memorable image is difficult, and to make things worse, the security people tell me that I need *different* passwords for *different* web sites. Now I'm expected to remember both "Gigantic Martian Insect Party" *and* "Structurally Unsound Yeti Tote-bag," and I have to somehow recall which phrase is associated with my banking web site, and which one is associated with some other site that doesn't involve extraterrestrial insects or Yeti accoutrements. This is uncivilized and I demand more from life. Thus, when security researchers tell me that they're not working on passwords, it's like physicists from World War II telling me that they're not working on radar or nuclear bombs, but instead they're unravelling the mystery of how bumblebees fly. It's like, you are so close, and yet so far. You almost get it, but that's worse than not getting it at all.

My point is that security people need to get their priorities straight. The "threat model" section of a security paper resembles the script for a telenovela that was written by a paranoid schizophrenic: there are elaborate narratives and grand conspiracy theories, and there are heroes and villains with fantastic (yet oddly constrained) powers that necessitate a grinding battle of emotional and technical attrition. In the real world, threat models are much simpler (see Figure 1). Basically, you're either dealing with Mossad or not-Mossad. If your adversary is not-Mossad, then you'll probably be fine if you pick a good password and don't respond to emails from ChEaPestPAiNPi11s@virus-basket.biz.ru. If your adversary *is* the Mossad, YOU'RE GONNA DIE AND THERE'S NOTHING THAT YOU CAN DO ABOUT IT. The Mossad is not intimidated by the fact that you employ https://. If the Mossad wants your data, they're going to use a drone to replace your cellphone with a piece of uranium that's shaped like a cellphone, and when you die of tumors filled with tumors, they're going to hold a press conference and say "It wasn't us" as they wear t-shirts that say "IT WAS DEFINITELY US," and then they're going to buy all of your stuff at your estate sale so that they can directly look at the photos of your vacation instead of reading your insipid emails about them. In summary, https:// and two dollars will get you a bus ticket to nowhere. Also, SANTA CLAUS ISN'T REAL. When it rains, it pours.

Threat	Ex-girlfriend/boyfriend breaking into your email account and publicly releasing your correspondence with the My Little Pony fan club	Organized criminals breaking into your email account and sending spam using your identity	The Mossad doing Mossad things with your email account
Solution	Strong passwords	Strong passwords + common sense (don't click on unsolicited herbal Viagra ads that result in keyloggers and sorrow)	◆ Magical amulets? ◆ Fake your own death, move into a submarine? ◆ YOU'RE STILL GONNA BE MOSSAD'ED UPON

Figure 1: Threat models

This World of Ours

The Mossad/not-Mossad duality is just one of the truths that security researchers try to hide from you. The security community employs a variety of misdirections and soothing words to obscure the ultimate nature of reality; in this regard, they resemble used car salesmen and Girl Scouts (whose "cookie sales" are merely shell companies for the Yakuza). When you read a security paper, there's often a sentence near the beginning that says "assume that a public key cryptosystem exists." The authors intend for you to read this sentence in a breezy, carefree way, as if establishing a scalable key infrastructure is a weekend project, akin to organizing a walk-in closet or taming a chinchilla. Given such a public key infrastructure, the authors propose all kinds of entertaining, Ferris Bueller-like things that you can do, like taking hashes of keys, and arranging keys into fanciful tree-like structures, and determining which users are bad so that their keys can be destroyed, or revoked, or mixed with concrete and rendered inert. To better describe the Mendelian genetics of keys, the authors will define kinky, unnatural operators for the keys, operators that are described as unholy by the Book of Leviticus and the state of Alabama, and whose definitions require you to parse opaque, subscript-based sentences like "Let K_R ₩ K_T represent the semi-Kasparov foo-dongle operation in a bipartite XY_{abc} space, such that the modulus is spilt but a new key is not made."

This Caligula-style key party sounds like great fun, but constructing a public key infrastructure is incredibly difficult in practice. When someone says "assume that a public key cryptosystem exists," this is roughly equivalent to saying "assume that you could clone dinosaurs, and that you could fill a park with these dinosaurs, and that you could get a ticket to this 'Jurassic Park,' and that you could stroll throughout this park without getting eaten, clawed, or otherwise quantum entangled with a macroscopic dinosaur particle." With public key cryptography, there's a horrible, fundamental challenge of finding somebody, *anybody*, to establish and maintain the infrastructure. For example, you could enlist a well-known technology company to do it, but this would offend the refined aesthetics of the vaguely Marxist but comfortably bourgeoisie hacker community who wants everything to be decentralized and who non-ironically believes that Tor is used for things besides drug deals and kidnapping plots. Alternatively, the public key infrastructure could use a decentralized "web-of-trust" model; in this architecture, individuals make their own keys and certify the keys of trusted associates, creating chains of attestation. "Chains of Attestation" is a great name for a heavy metal band, but it is less practical in the real, non-Ozzy-Ozbourne-based world, since I don't just need a chain of attestation between me and some unknown, filthy stranger —I also need a chain of attestation *for each link in that chain*. This recursive attestation eventually leads to fractals and H.P. Lovecraft-style madness. Web-of-trust cryptosystems also result in the generation of emails with incredibly short bodies (e.g., "R U gonna be at the gym 2nite?!?!?!?") and multi-kilobyte PGP key attachments, leading to a packet framing overhead of 98.5%. PGP enthusiasts are like your friend with the ethno-literature degree whose multi-paragraph email signature has fourteen Buddhist quotes about wisdom and mankind's relationship to trees. It's like, I GET IT. You care deeply about the things that you care about. Please leave me alone so that I can ponder the inevitability of death.

Even worse than the PGP acolytes are the folks who claim that we can use online social networks to bootstrap a key infrastructure. Sadly, the people in an online social network are the same confused, ill-equipped blunderhats who inhabit the physical world. Thus, social network people are the same people who install desktop search toolbars, and who try to click on the monkey to win an iPad, and who are willing to at least entertain the notion that buying a fortune-telling app for any more money than "no money" is a good idea. These are not the best people in the history of people, yet somehow, I am supposed to stitch these clowns into a rich cryptographic tapestry that supports key revocation and verifiable audit trails. One time, I was on a plane, and a man asked me why his laptop wasn't working, and I tried to hit the power button, and I noticed that the power button was sticky, and I said, hey, why is the power button sticky, and he said, oh, IT'S BECAUSE I SPILLED AN ENTIRE SODA ONTO IT BUT THAT'S NOT A PROBLEM RIGHT? I don't think that this dude is ready to orchestrate cryptographic operations on 2048-bit integers.

Another myth spread by security researchers is that the planet Earth contains more than six programmers who can correctly use security labels and information flow control (IFC). This belief requires one to assume that, even though the most popular variable names are "thing" and "thing2," programmers will magically become disciplined software architects when confronted with a Dungeons-and-Dragons-style type system that requires variables to be annotated with rich biographical data and a list of vulnerabilities to output sinks. People feel genuine anxiety when asked if they want large fries for just 50 cents more, so I doubt that unfathomable lattice-based calculus is going to be a hit with the youths. I mean, yes, I understand how one can use labels to write a secure version of HelloWorld(), but once my program gets bigger than ten functions, my desire to think about combinatorial label flows will decrease and be replaced by an urgent desire to DECLASSIFY() so that I can go home and stop worrying about morally troubling phrases like "taint explosion" that are typically associated with the diaper industry and FEMA. I realize that, in an ideal world, I would recycle my trash, and contribute 10% of my income to charity, and willingly accept the cognitive overhead of fine-grained security labels. However, pragmatists understand that

This World of Ours

I will spend the bulk of my disposable income on comic books, and instead of recycling, I will throw all of my trash into New Jersey, where it will self-organize into elaborate "Matrix"-like simulations of the seagull world, simulations that consist solely of choking-hazard-sized particles and seagull-shaped objects that are not seagulls and that will not respond to seagull mating rituals by producing new seagull children. This is definitely a problem, but problem identification is what makes science fun, and now we know that we need to send SWAT teams into New Jersey to disarm a trash-based cellular automaton that threatens the seagull way of life. Similarly, we know that IFC research should not focus on what would happen if I somehow used seventeen types of labels to describe three types of variables. Instead, IFC research should focus on what will happen when I definitely give all my variables The God Label so that my program compiles and I can return to my loved ones. [Incidentally, I think that "The God Label" was an important plot device in the sixth "Dune" novel, but I stopped reading that series after the fifth book and my seven-hundredth time reading a speech that started "WHOEVER CONTROLS THE SPICE CONTROLS THE (SOME THING WHICH IS NOT THE SPICE)." Also note that if a police officer ever tries to give you a speeding ticket, do *not* tell him that you are the Kwisatz Haderach and You Can See Where No Bene Gesserit Can See and you cannot see a speeding ticket. This defense will not hold up in court, and the only "spice" that you will find in prison is made of mouthwash and fermented oranges.]

The worst part about growing up is that the world becomes more constrained. As a child, it seems completely reasonable to build a spaceship out of bed sheets, firecrackers, and lawn furniture; as you get older, you realize that the S.S. Improbable will not take you to space, but instead a lonely killing field of fire, Child Protective Services, and awkward local news interviews, not necessarily in that order, but with everything showing up eventually. Security research is the continual process of discovering that your spaceship is a deathtrap. However, as John F. Kennedy once said, "SCREW IT WE'RE GOING TO THE MOON." I cannot live my life in fear because someone named PhreakusMaximus at DefConHat 2014 showed that you can induce peanut allergies at a distance using an SMS message and a lock of your victim's hair. If that's how it is, I accept it and move on. Thinking about security is like thinking about where to ride your motorcycle: the safe places are no fun, and the fun places are not safe. I shall ride wherever my spirit takes me, and I shall find my Gigantic Martian Insect Party, and I will, uh, probably be rent asunder by huge cryptozoological mandibles, but I will die like Thomas Jefferson: free, defiant, and without a security label.

Why Join USENIX?

OPEN ACCESS

We support members' professional and technical development through many ongoing activities, including:

- Open access to research presented at our events
- Workshops on hot topics
- Conferences presenting the latest in research and practice
- LISA: The USENIX Special Interest Group for Sysadmins
- *;login:*, the magazine of USENIX
- Student outreach

Your membership dollars go towards programs including:

- Open access policy: All conference papers and videos are immediately free to everyone upon publication
- Student program, including grants for conference attendance
- Good Works program

Helping our many communities share, develop, and adopt ground-breaking ideas in advanced technology

Join us at www.usenix.org

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION