DECEMBER 2009 VOLUME 34 NUMBER 6

;login:

THE USENIX MAGAZINE

USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events

First Workshop on Sustainable Information Technology (SustainIT '10)

Co-located with FAST '10

FEBRUARY 22, 2010, SAN JOSE, CA, USA
http://www.usenix.org/sustainit10

2nd USENIX Workshop on the Theory and Practice of Provenance (TaPP '10)

Co-located with FAST '10

FEBRUARY 22, 2010, SAN JOSE, CA, USA
http://www.usenix.org/tapp10
Submissions due: December 14, 2009

8th USENIX Conference on File and Storage Technologies (FAST '10)

Sponsored by USENIX in cooperation with ACM SIGOPS

FEBRUARY 23–26, 2010, SAN JOSE, CA, USA
http://www.usenix.org/fast10

3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '10)

Co-located with NSDI '10

APRIL 27, 2010, SAN JOSE, CA, USA
http://www.usenix.org/leet10
Submissions due: February 25, 2010

2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN '10)

Co-located with NSDI '10

APRIL 27, 2010, SAN JOSE, CA, USA
http://www.usenix.org/inmwren10
Paper registration due: February 5, 2010

9th International Workshop on Peer-to-Peer Systems (IPTPS '10)

Co-located with NSDI '10

APRIL 27, 2010, SAN JOSE, CA, USA
http://www.usenix.org/iptps10
Submissions due: December 18, 2009

7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

APRIL 28–30, 2010, SAN JOSE, CA, USA
http://www.usenix.org/nsdi10

2nd USENIX Workshop on Hot Topics in Parallelism (HotPar '10)

Sponsored by USENIX in cooperation with ACM SIGMETRICS, ACM SIGSOFT, ACM SIGOPS, and ACM SIGARCH

JUNE 14–15, 2010, BERKELEY, CA, USA
http://www.usenix.org/hotpar10
Submissions due: January 24, 2010

8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys 2010)

Jointly sponsored by ACM SIGMOBILE and the USENIX Association

JUNE 14–18, 2010, SAN FRANCISCO, CA, USA
http://www.sigmobile.org/mobisys/2010/
Abstracts due: December 4, 2009

USENIX Technical Conferences Week

2010 USENIX Annual Technical Conference (USENIX ATC '10)

JUNE 23–25, 2010, BOSTON, MA, USA
http://www.usenix.org/atc10
Submissions due: January 11, 2010

USENIX Conference on Web Application Development (WebApps '10)

JUNE 23–25, 2010, BOSTON, MA, USA
http://www.usenix.org/webapps10
Paper titles and abstracts due: January 4, 2010

3rd Workshop on Online Social Networks (WOSN 2010)

JUNE 22, 2010, BOSTON, MA, USA
http://www.usenix.org/wosn10
Paper submissions due: February 18, 2010

2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)

2nd Workshop on Hot Topics in Storage and File Systems (HotStorage '10)

19th USENIX Security Symposium (USENIX Security '10)

AUGUST 11–13, 2010, WASHINGTON, DC, USA
http://www.usenix.org/sec10
Submissions due: February 5, 2010

For a complete list of all USENIX & USENIX co-sponsored events, see http://www.usenix.org/events.

contents

;login:

VOL. 34, #6, DECEMBER 2009

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Casey Henderson
Jane-Ellen Long
Jennifer Peterson

TYPESETTER
Star Type
startype@comcast.net

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

http://www.usenix.org
http://www.sage.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.

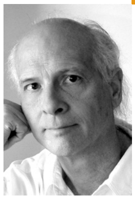
$90 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $125 per year.

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.

RIK FARROW

musings



Rik is the Editor of *;login:*.

rik@usenix.org

IMAGINE WATCHING THE BEGINNING of a stock car race. As the drivers climb into their cars, they ignore the webbing for covering the drivers' windows and do not attach their five-point restraints or the head-and-neck supports. Although this goes against safety recommendations (and NASCAR rules), the drivers have decided that these safety measures are "inconvenient" and "interfere with the experience."

Certainly my imagined event sounds unbelievable today. Yet at the end of Jeremiah Grossman's invited talk at the 2009 USENIX Security Symposium, I asked the audience, over 300 of the people who had chosen to attend the premier security research conference, for a show of hands on users of NoScript. Only a few hands went up, and I sat down, astonished.

Later that evening, a usability researcher approached me. He said that NoScript was considered a bad example of usability. I certainly understand that, yet when the consequences of not using NoScript are considered, it is like not choosing to wear a seatbelt while racing because of the inconvenience. That security researchers would choose a "better experience" and "convenience" over Web browsing safety still amazes me.

Through the Browser Window

For many years now, Web browsers have been the pen testers' choice for getting past firewalls. I know some very good penetration testers, and as firewalls became common as well as including better configurations, the initial penetration method was to attack via Web browsers. The pen testers may abuse vulnerabilities in Web browsers, but just as often they simply use features of Web browsers combined with normal human weaknesses, such as trust, that are easily exploited. While home users' Windows systems are the most infected with malware installed via driveby attacks [1], businesses are not immune. Businesses can be targeted using techniques similar to those of my pen testing friends, but the goals in these cases are different. Businesses may be targeted for intellectual property or secrets, but these days the target is often bank account information [2, 3].

I did ask Jeremiah Grossman if he used NoScript. Grossman said he did, which is not surprising considering that he had just presented the Top 10 Web

Hacking Techniques (see article on p. 16) and NoScript blocks several of these attacks. I particularly shuddered when contemplating clickjacking, a technique that allows an attacker to trick a browser user into clicking on the button of the attacker's choice. Clickjacking is a feature of modern browsers, allowing an attacker to move (hover) over an iframe so that the button to be clicked is always under the user's mouse. And this happens invisibly to the user, as the iframe is hidden beneath other content.

Many years ago, I mused about having a button on my browser that would give me the option of allowing scripting to work for a particular site. NoScript has provided that button for many years now, and Giorgio Maone has continued to add security features to NoScript over the years as well. I asked Maone if he wanted to write about NoScript for this issue, and you can find his article on p. 21.

Maone considers the same usability issues that some people complain about to be a feature. He wants NoScript to work without cluttering up your browser with popups. I will say that I had to learn that when a Web site doesn't work as expected, I need to see what NoScript is blocking. For the most part, I have already whitelisted the sites I trust, which also happen to be those I visit often. When I visit a new site, I have to decide whether to allow scripting to work. I usually enable scripting temporarily unless I know I will be visiting a site frequently. And I don't enable scripting for all sites that request the ability, as some of these sites just use scripting to present advertising or to collect information about your browsing habits. I'd rather maintain my privacy. And when I want to buy something, I will research products rather than buy the product with the spiffy and/or annoying ad.

Advertising sites themselves can be sources of malware. In [1], the researchers mention that a source of drive-by downloads comes from reselling advertising slots on Web pages. If there are no current buyers for advertising on a particular Web page, these potential slots can be resold to other advertising networks. You could wind up being the victim of an attack even when visiting a trusted site, if you don't use NoScript. Because NoScript blocks scripting based on the site the script comes from, you can still view your favorite site while preventing other sites from executing scripts.

Banks and Credit Cards

Credit card companies provide you with some insulation against loss of credentials. If someone steals your credit card info, you can report it to the credit card company and pay a limited amount (at most $50 in the US). Banks, however, look at credential loss completely differently. Banks have traditionally focused on using SSL to protect transaction data while it traverses the Internet. At the same time, banks assume that the endpoints of the communication, including browsers, are secure. Yet that is unlikely to be the case today for most PC users.

Even the use of one-time passwords and password generation tokens does not provide protection for users of malware-infected PCs. Malware has been designed to wait until the user has provided authentication and then to initiate a fund transfer request that appears to the bank to be authenticated. If you can't trust your own computer, SSL really doesn't help you at all. Adding insult to injury, banks in some countries hold users responsible for losses if the computers they use are not secure.

In the US, recent looting of the bank accounts of small businesses and even a county have garnered some news. And in these cases, the owners of the accounts, whose credentials have been stolen using malware and botnets, were held responsible for the losses as well.

Secure Operating Systems

It would be helpful if we could use secure operating systems. Just recently, an Australian research project undertaken to prove the correctness of an operating system, the seL4 microkernel, was completed. You can read what one of the researchers, Gerwin Klein, has to say about this starting on p 28 of this issue. The operating system executes with the highest level of privilege and has exclusive access to all hardware devices, including disk and network devices, as well as arranging for access to pages in memory. Having an operating system with proven security guarantees is a great leap forward.

We also need secure applications that we can use. I've written about the OP browser before [4], a browser which uses process-based isolation for each site that goes well beyond what Google Chrome and IE8 do today. Just the week before I completed this column, the source code to the OP browser went online [5]. There are still issues with the OP browser, mainly having to do with running isolated windows on top of window managers that do not support the concept of sharing a display among different security principles—that is, each site acting as a separate user, isolated from what other sites can do. And sites that rely on overlapping views, such as mashups that use overlays on top of maps, are very difficult to deal with. But the OP browser, because of its design, already shows higher performance on multicore systems than IE8 or Chrome for certain tasks.

SECURE TCP

Not even TCP itself can be considered safe. Security issues with TCP connection state have been known since 1985 [6], but little has been done beyond quick fixes, such as initial sequence number randomization. As DNSSEC begins to see wider adoption, starting in December 2009, root server operators will really be feeling the effects of having to support TCP connections, as TCP connection state can easily be abused. Attackers began using SYN floods against TCP in 1996, and only non-standard kludges defend against these and similar denial-of-service attacks today.

Metzger, Simpson, and Vixie have written about a change to TCP that eliminates these issues. This change, TCP Cookie Transactions (TCPCT), has been discussed for many years in some form and appears close to being implemented in at least two OS stacks soon. TCPCT can easily be integrated into the Internet, as the new option will be ignored by systems, including firewalls, that don't recognize it. You can read about TCP Cookie Transactions beginning on p. 35 of this issue.

Lineup

I have actually touched on many of the articles in this issue of *;login:* already. Dave Dittrich has written an article that both recalls the history of distributed denial of service (DDoS) attacks and ethics. Both researchers and investigators need to be bound by a code of ethics, perhaps legally bound. Dittrich carefully covers this concept with a story about how he collected the source code to early DDoS tools.

Peter Galvin covers an emerging feature in OpenSolaris: Immutable Service Containers (ISCs). ISCs are a containment mechanism designed to be used for networked services. Initially they work with Solaris Zones, but may eventually work under Solaris VM environments as well. ISCs promise to be another useful tool for securing services.

David Blank-Edelman provides more general advice for Perl programming, or, as he has put it, he "likes to get meta." Blank-Edelman describes simple techniques, as well as how to endure them, for improving the robustness of your Perl code.

Dave Josephsen provides 7 tips for successful Nagios implementations. Actually, you would do well to pay attention to his list no matter what type of monitoring and reporting you are doing.

Robert Ferrell regales us with his own definitions of terms used in the security industry. Robert has his own way of looking at things, as you will have noticed. I find that I can strongly agree with Robert on his definitions, as a large dose of cynicism is in order when it comes to computer security.

We have lots of book reviews this time around, and we close with reports from the 2009 USENIX Security Symposium and two associated workshops: HotSec and CSET.

I became paranoid about UNIX security, and later Internet security, starting in 1984. That was the year someone shared a much-copied list of security exploits that had occurred at UCSC over a few years. The list provided a reminder of what clever students could do with a little knowledge and a dose of misguided motivation.

Today, exploiting browsers is big business. Exploits are sold on the black market, converted into easy-to-use toolkits for exploiting browsers and Web servers, then sold. These tools are designed to steal login credentials or to proxy authenticated connections to banks and financial institutions. Nothing magical is involved here, as our current Web browser technologies actually support the installation and use of tools that have browser-wide impact. In fact, without this support, NoScript itself would not work.

Strap into your Web browsers! I encourage you to endure the inconvenience of having to decide, perhaps after some research, whether you consider a site safe or not. While NoScript's user interface could be easier to use, I find a bit of inconvenience a lot more palatable than the consequences.

REFERENCES

[1] N. Provos, P. Mavrommatis, M.A. Rajab, and F. Monrose, "All Your iFRAMEs Point to Us," *Proceedings of the 17th USENIX Security Symposium*, July 2008, pp. 1–15.

[2] Kelly Jackson Higgins, "Attack of the Mini-Botnets," DarkReading: http://www.darkreading.com/security/attacks/showArticle.jhtml?articleID=216402026.

[3] "Clampi Targets Banking Info": http://www.usatoday.com/tech/news/computersecurity/2009-07-30-clampi-computer-virus_N.htm.

[4] Chris Grier, Shuo Tang, and Samuel T. King, "Building a More Secure Web Browser": http://www.usenix.org/publications/login/2008-08/pdfs/grier.pdf; Rik Farrow, "Musings," http://www.usenix.org/publications/login/2008-08/openpdfs/musings.pdf.

[5] OP-Browser source: http://code.google.com/p/op-web-browser/source/checkout.

[6] R. Morris, "A Weakness in the 4.2 BSD UNIX TCP/IP Software": pdos.csail.mit.edu/~rtm/papers/117.pdf.

Thanks to USENIX and SAGE Corporate Supporters

USENIX Patrons
Google
Microsoft Research

USENIX Benefactors
Hewlett-Packard
IBM
Infosys
Linux Pro Magazine
NetApp
Sun Microsystems
VMware

USENIX & SAGE Partners
Ajava Systems, Inc.
BigFix
DigiCert® SSL Certification
FOTO SEARCH Stock Footage and Stock Photography
Splunk
SpringSource
Zenoss

USENIX Partners
Cambridge Computer Services, Inc.
GroundWork Open Source Solutions
Xirrus

SAGE Partner
MSB Associates

DAVID DITTRICH

the conflicts facing those responding to cyberconflict

Dave Dittrich holds an appointment as an affiliate principal scientist with the Applied Physics Lab at the University of Washington. He has studied distributed denial of service (DDoS) attack tools, botnets, host and network forensics, and the legal/ethical framework for responding to computer attack (the "Active Response Continuum") for over a decade.

dittrich@speakeasy.net

THERE IS INCREASING TALK OF CYBERwarfare, where the computers of private citizens are the weapons being used. Out of a sense of frustration, some call for a right to self-defense and for going on the offensive against cyberattackers. Researchers today are regularly doing things that cause effects visible to others and publishing information under the banner of full disclosure without supporting their decisions through a systematic analysis of both the legal and the ethical issues involved. We are entering a dangerous time and have a lot to talk about and agree upon, lest someone with good intentions causes massive harm.

The purpose of this article is twofold. First, I want to encourage the computer security community to discuss how ethics apply to responses to cyberconflict. Second, I want those outside the computer security community to understand how much more damaging and serious the situation is becoming. Such an understanding is necessary to help get the policy and legal changes necessary to address today's increased threat landscape in ways that are acceptable to society.

I use my own story here as a case study in how some of these issues are raised and how they can be addressed, centering on events that led to the first distributed denial of service (DDoS) attacks over a decade ago. Anyone reading my analysis of the trinoo distributed denial of service attack tool written in October 1999 and released to the public on December 30, 1999, would have noticed the following:

> Trinoo daemons were originally found in binary form on a number of Solaris 2.x systems, which were identified as having been compromised by exploitation of buffer overrun bugs in the RPC services "statd", "cmsd" and "ttdbserverd". These attacks are described in CERT Incident Note 99-04:
>
> http://www.cert.org/incident_notes/IN-99-04.html
>
> The trinoo daemons were originally believed to be UDP based, access-restricted remote command shells, possibly used in conjunction with sniffers to automate recovering sniffer logs.
>
> During investigation of these intrusions, the installation of a trinoo network was caught in the act and the trinoo source code was obtained

> from the account used to cache the intruders' tools and log files. This analysis was done using this recovered source code.

These statements play down the significance of distributed intruder attack tools, an advance that was taking place beyond the gaze of the public. Attacks have become more automated, more sophisticated, and more complex. This has put great pressure on incident responders to deal with the increase in abuse and compromised systems. Incident response teams face a choice. They can take the easy route, wiping and re-installing systems and spending just enough effort to keep up with the onslaught. Of course this "easy way out" makes it harder for law enforcement to do their job, possibly resulting in more harm to society (a concept known as *externalizing costs*). Or they can make the effort to find ways to be more efficient, effective, and proactive at countering cybercrime. This might mean taking aggressive actions to home in on crucial evidence for attributing criminal acts, or identifying key attacker assets (e.g., command and control servers) and finding ways of taking those assets out of the hands of attackers to neutralize their ability to cause harm to society. The latter option has its own potential risks of harm to society—starting with loss of privacy, but ranging up to possible disruption, destruction of computer data, or even physical damage—which can be mitigated through ethical decision-making that systematically balances potential harms and benefits.

The Advent of Distributed Attacks

Distributed denial of service (DDoS) became widely known when high profile targets such as Yahoo, CNN, Amazon, and eBay were attacked in February 2000. Denial of service itself was not new, but the remote control of thousands of computers at a time was. And it started many months earlier than the public knew. What had once been manual and limited to the number of people who could type on command lines became automated, distributed, and allowed a handful of malicious actors to create orders of magnitude more damage than before.

SNIFFER ATTACKS

In the mid to late 1990s, computer intrusions involving the installation of *sniffers*, programs that monitor network traffic for the purpose of stealing login credentials, were rampant. Many sites were still using older TCP/IP protocols, such as telnet, ftp, imap, pop, and rlogin, for remote terminal sessions, file transfers, and email. The problem was that networks at the time were often wired using *thin-wire Ethernet*, a shared network medium on which any host was capable of seeing all network traffic to/from any other host. Login names and passwords were visible to anyone who could control a computer on the same network segment. The result at large universities was massive account theft and abuse. Someone possessing a list of several hundred stolen accounts could hop from account to account, remaining active within the network for over a year. Intrusions also spread quickly from one host to many other hosts and many other sites. There was still a limiting factor: attackers had to manually install sniffers and come back later to retrieve the sniffer logs. The solution: client-server computing!

DISTRIBUTED SNIFFERS

In the early months of 1999, an attentive system programmer, responsible for the large clusters of IBM AIX systems that the University of Washington (UW) made available centrally, noticed some odd processes that showed very long up times. He had wisely obtained process memory dumps before killing the processes and made note of listening network ports that indicated that the program might be a remote access trojan.

I analyzed the memory dumps and identified what looked like a simple array data structure with a user name, a password, and a numeric value. I identified the numeric values as UNIX timestamps. Comparison of last login records showed a correlation between the account names and timestamps. Analysis of process lists and login records on other cluster members for the accounts involved showed that at one point, over a month earlier, someone had started running the same program on all hosts in the cluster. What we had discovered was the first known distributed sniffer for IBM AIX systems. Now the sniffer logs from all hosts in the cluster could be automatically retrieved over the Internet in rapid succession, increasing the scale and speed of credential harvesting.

While a single host on a shared Ethernet segment could net a few hundred login credentials in a month, having a local sniffer on every host in a large cluster used by tens of thousands of people could potentially yield orders of magnitude more. This was not the only type of attack being automated, though.

DISTRIBUTED DOS TOOLS

Denial of service attacks in the late 1990s relied on using stolen accounts to run programs such as synk4, teardrop, or smurf from the command line. The first two programs implemented point-to-point attacks, where bandwidth and the number of accounts used dictated who would win. A *smurf attack* was a form of reflected and amplified DoS attack that exploited poorly configured network routers and required far fewer accounts to initiate the attacks. Regardless, such floods were straightforward to identify and trace back to the accounts used to run these attack programs, which could then be easily shut down. The primary limiting factor was the number of attackers who could log into stolen accounts, download DoS attack programs, and initiate attacks.

Things changed significantly in the summer of 1999. UW began to get reports of DoS attacks against different sites around the world, all involving dozens of UW systems *all at the same time*. Some unknown program was being installed and run on dozens of compromised computers. It was controlled through remote connections from a small handful of central locations, using an unknown protocol. It looked similar to the distributed sniffers and had the same problem of being indirectly controlled. And it was capable of flooding remote hosts in several different ways, keeping them offline for days at a time.

Nobody had ever dealt with this kind of distributed attack before. Nobody knew exactly how it worked. Worst of all, nobody knew how to stop it. We desperately needed a detailed understanding of how to identify these distributed denial of service programs on infected computers, how to identify them by observing network traffic patterns, and how to scan our networks to quickly find infected hosts and get them cleaned up. Our responses had to scale as well as these new attacks, both in social and technical terms.

My Story

Responding to account abuse involves determining whether the account holder knowingly misused their account or gave their password to someone who abused the account (a policy violation), or unknowingly had their login credentials stolen by an outsider (i.e., they are innocent). One method of investigating these attacks involves examination of account contents, including files stored in the account and saved email messages. To respect the privacy of the user, while learning who was responsible for account abuse, I adopted an investigative method, patterned on the FBI's procedure used when wiretapping, known as *minimization*. This means starting with the least invasive methods first and only using more invasive analysis if evidence warranted it. Searching for keywords like *password*, *pwd*, *pw:*, *account*, *acct*, *acc:*, etc. could indicate purposeful sharing. Only if I found such keywords would I then expand the search to include the email headers for Subject, To, Cc, From, Sender, Date, etc. to provide more context. Finally, I would only look at the specific messages (identified by the header lines) and the specific paragraph in which the keywords occurred. If it looked like that portion of message had nothing to do with account abuse, I would immediately stop reading and go on to the next suspect message. My task was only to verify account sharing, not read personal communications. This conformed with policies for protecting UW systems, maximizing efforts to secure UW systems, and minimizing intrusion into account holder privacy.

The same minimization techniques can be applied to analysis of the content of files stored in suspect accounts. By correlating login history with the creation or modification dates of files in the account and looking at their names, it was possible to identify those files that were created during periods of suspected abuse (e.g., a DoS attack, spam run, or scanning activity). It was not necessary to wade through any/all files, which would be more invasive to privacy. Correlating this information across multiple abused accounts often illuminated a pattern. One host, or one domain name, might show up as the source for logins across multiple abused accounts. By looking at the exploit programs stored in the stolen accounts, it was sometimes possible to identify a specific target (e.g., Linux running imapd). A quick scan of the suspect network for any hosts with the vulnerability being used by the attackers often allowed me to locate the host running the sniffer. I would then target that host for forensic analysis.

I wrote scripts that parsed the log files produced by several of the most commonly used malicious sniffers. These sniffer logs typically showed the source and destination hosts, the login account name, what protocol or service was involved, and the first couple of dozen to couple of hundred characters typed. This latter text is where the password is found and sometimes also the first few commands that were typed (e.g., logins to other hosts, the root password in `su` or `sudo` commands). This script allowed me to extract a list of all accounts that were compromised by the sniffer and on which systems those accounts existed.

I modified another script, originally written by a brilliant programmer at UW named Corey Satten, to extract lines by domain name, IP address, and even network block in Classless Internet Domain Routing (CIDR) notation. I could process the compromised account and host lists from the sniffer logs and split them up by (1) those at UW and (2) those at remote sites. Another script would iterate over the list of remote sites and send one email per site reporting those accounts or hosts at just that site. Rather than sending one big list to everyone, which would expose information about all victims, I could do targeted reporting. This allowed the efficient reporting of compromised assets at *all sites affected at one time* and I could proactively identify and remove from attacker control *all compromised resources at one time* instead of waiting for abuse reports to come in and spending far more time handling them individually. While this kind of response was more complex, on several occasions it resulted in distant hackers leaving the UW network and not coming back. This was fighting automation of attacks with automation of defensive response. The technique was quite effective, at least in those cases where affected sites were capable and cooperative in removing access to malicious hackers.

During the routine investigation of a suspected sniffer on a UW subnet in the summer of 1999, I was able to locate the sniffer on a Linux workstation, retrieve its log file, and began extracting account/host information to do the all-at-once cleanup. One entry in the sniffer log caught my eye, however. It showed a connection from a *different* host on the same shared network segment, but sourced from a computer owned by a completely different research group. Such shared networks were the easiest way for an intrusion to spread quickly. Effectively, one hacker's sniffer had managed to capture a connection exposing *another hacker's* activity! I recognized the name of the second computer as one of the Sun computers involved in DDoS attacks weeks earlier. The sniffer had been running long enough to capture historical evidence of the installation of a DDoS agent. I now knew where the DDoS attackers were caching their trinoo agent binaries (and possibly much more, given recent experience). Had I simply reported the sniffer to the owner, who would have been tempted to just wipe and reinstall the OS, this key piece of information would have disappeared forever.

Recognizing the significance of what I had learned, I quickly put on my headset, looked up the technical contact information from the relevant domain registry and initiated a telephone call. My intention was to (1) contact a responsible party at the affected site to report the intrusion into their network and (2) request that they preserve evidence and provide me with a copy of files in the directory to analyze. While they might not (and odds were good they probably didn't) have a skilled incident response team, with someone who understood computer attack tools, networking, programming, and scripting, I was prepared to make use of the information to try to put a stop to the harm being caused to systems around the globe. If this company didn't report the incident to law enforcement, it meant that the investigation hit a dead end. If they did report it to law enforcement, it might take weeks or months (if ever) to complete a detailed analysis. Even if such analysis was performed, it might not be widely enough distributed to achieve worldwide mitigation of the event. The likelihood was that if I did not get my hands on the data in that account, key information would simply disappear and the damage worldwide would continue to escalate.

When making blind contact with remote sites in situations like this, in the middle of active hostile activity, there are many common outcomes I have encountered over the years. Most of the time, explaining who I am and what I am doing results in cooperation by the site being contacted. Sometimes they even offer to provide a root password and let me clean up the system, which I decline and instead provide guidance to help them clean up their own systems. Sometimes they say, "Thanks for reporting this to us," and immediately take the system off-line, wipe the hard drive, and reinstall the operating system. This destroys most/all evidence on the system. They may not even report to law enforcement, which results in a dead end for investigation. Sometimes the person denies there could possibly be any problem or gets mad and hangs up. Sometimes they are distrustful and assume I am somehow involved. In a handful of cases, the person I spoke with professed to be helping to stop the intruder, even asking that I contact them immediately whenever I noticed this person using their systems. They later turned out to be actively helping the intruder, or they themselves *were* the intruder. Sometimes my report gets sent around the organization in email which happens to be actively monitored by the bad guys.

I reached the operator at the victim site and informed them I had reason to believe that one of their main computer servers had been compromised and that I needed to speak with someone responsible for computer security investigations at their company. My call was transferred. The conversation began something like this:

> Hello. My name is David Dittrich and I am a computer security engineer at the University of Washington in Seattle. If you wish to verify my identity, you can call the main switchboard at the University of Washington and ask to be transferred to me, or get my contact information from my Web site, which you can find with a search engine by entering my name. I understand if you don't trust what I am saying to you.
>
> I am investigating a series of intrusions at the University of Washington that involve distributed denial of service attacks involving thousands of computers around the world. One attack last month disrupted the entire campus of the University of Minnesota for over three days. These attacks have been reported to the FBI and are under active investigation. I have a report that I can send you that details these attacks so you can understand their complexity and impact.
>
> I have evidence that your computer system *hostname* has been compromised and is actively being used by someone for engaging in remote attacks against systems around the globe. Your system holds files associated with these attacks that are central to understanding them and trying to identify who is perpetrating them. I urge you to report this intrusion to the RCMP.
>
> I am requesting your permission to analyze the files contained in the compromised account I have identified and I advise that you make your own bit-image copy of the hard drive to preserve evidence that may still exist in unused file space.

The response I got was positive. The person verified who I was by looking at my Web site and said they appreciated the call and wanted to help. I was granted the permission I requested on the condition that I promised (1) to give them full details of how their system was compromised and how it was being used by the attackers, (2) never to disclose the name of their company and (3) not to publish any corporate or customer data I might encounter that was unrelated to the illegal activity of the attackers. I have to this day adhered to and will continue to adhere to all aspects of this promise.

The fruits of my analysis were the first detailed technical understanding of distributed denial of service attack tools. I produced details of how to detect these programs on infected hosts, how to detect them on the wire, and how to scan for them remotely. Instructions and guidance on how to locally scan one's own network were given, along with cautions about likely countermeasures that could result in false-negative checks. I circulated these analyses privately at first, trying to provide as much lead time as possible for law enforcement, the military, policymakers, and the security industry to prepare for what might come next. My analyses were used as discussion points for the first workshop ever organized and sponsored by CERT/CC [9]. Several years later I co-authored the first book on Internet denial of service attacks with Jelena Mirkovic, Peter Reiher, and Sven Dietrich [6].

In the years following the release of the initial DDoS attack tool analyses, those publications were widely cited in academic research as among the first references on the subject. A new class of security products and services designed to detect and mitigate DDoS attacks was also created, many starting out by addressing the specifics detailed in these same analyses. However, the technical details of these attack tools alone were not the only insights into the complex nature of responding to large-scale distributed attacks. DDoS attacks, distributed tools, and *botnets* (as they are now popularly known) are multi-phase attacks that involve a complex arrangement of compromised resources spread across networks around the globe and organized into a coordinated attack network. There is much more to countering these threats than just detecting and blocking a flood of packets, and it demands similar automation of response actions, coordination between involved sites, and a deep knowledge of the attackers' tools and tactics.

Responding to Complex Computer Network Attacks

The issues of responding to increasingly sophisticated and complex computer network attacks that are illustrated here are not new. Just a few years earlier, President Clinton created the President's Commission on Critical Infrastructure Protection (PCCIP) to advise his administration on how to deal with the threats to critical infrastructures that were emerging from widespread Internet connectivity. The PCCIP produced a series of reports, entitled the *Legal Foundations Study* [8], which addressed such issues as difficulties in detecting computer crime, resource constraints, the existing legal landscape, (in)adequacy of existing criminal statutes, strains on federal law enforcement investigation and prosecution capabilities, international agreements on cooperation in cyber-investigations, and proposed new approaches to enhancing cyber-intrusion response. Two of the principal authors of the PCCIP reports published a law review article in which they call for a balanced public/private approach for responding to cybercrime that includes oversight mechanisms such as licensing and certification [7].

There have been other discussions of these topics in the private sector [1, 2]. Kenneth Einar Himma and I co-authored an article on what I am calling the *active response continuum* (ARC) in which some of the legal and ethical issues are raised [5]. We describe the issues in responding to large-scale coordinated attacks in the face of differences in skill level and capacity of the victim sites involved and other issues brought up by the PCCIP reports. I prefer the term "active response continuum" over "active defense," to stress the range from low to high of the capacity to respond, aggressiveness of actions, and risk of harm that must be balanced against intended benefit. It is very common for discussions involving people new to this topic, who have never engaged in coordinated and collaborative response to computer intrusions, to jump to simplistic self-defense analogies and call for the right to "hack-back" or "counter-strike." This is both naïve and very risky, as these are at one extreme end of the spectrum. Arguing simply whether or not one has a right to "hack-back" in self-defense misses more subtle, and less risky, alternatives. Similarly, all the various options along the continuum are not viewed in relation to effects on others who are simultaneously investigating and responding to the same widespread events, or those using the computer systems and networks involved in criminal activity.

Conclusion

Times have changed since 1999. The days of finding the source code, log files, exploit tools, etc., being cached in one place for months at a time are quite rare for the most advanced attacks. The sophisticated attacker knows better and does a much better job of operational security. This requires a more sophisticated response with more difficult challenges to overcome than in years past. Law enforcement is now far more coordinated internationally, more highly staffed, and more engaged with groups I will call *mitigation communities* whose good intentions and talents are applied to counter today's sophisticated cybercrime.

Good intention alone is not sufficient in deciding whether or not to take aggressive or risky actions in response to cybercrime, or in choosing which action to take. There are a host of unintended consequences that result from one's actions. It is important to have as much knowledge as possible about the behavior of attackers and the capabilities of their tools. It is hard enough to reverse engineer sophisticated malware, but finding an attacker's weakness and immediately leaping to disclose it or attack it is unwise in the extreme. It is equally hard to develop a sophisticated counterattack that considers the *effects* of any action one might take, in terms of benefit or advantage as well as risk or harm (e.g., privacy violation.)

This is not a situation where one group of white hats congregating in an online vetted community goes toe-to-toe with another group of black hats who congregate in their underground equivalent. There are millions of innocent third parties standing between and around us who just want to go about their daily business, using the Internet to enhance their lives. They don't want to be harmed by getting caught in the cyber cross-fire. The effect of a mistake that causes widespread harm to the general public could be significant, resulting in a public-opinion backlash or knee-jerk legislation that significantly sets back the efforts of defenders and puts attackers in an even stronger position. Solidifying the gap between government agencies and the private sector, or allowing researchers to perform crime-scene-altering experiments in an uncontrolled and uncoordinated manner, will similarly prevent a comprehensive cyber-response capability and further the damage currently being done to our nation.

My colleagues and I presented a poster at the 16th ACM Conference on Computer and Communications Security [3] that is based on a technical report [4] in which we call for a structured debate of the ethical issues surrounding computer security research activities that will guide decision-making in a more sophisticated and deliberate manner. This technical report contains over two dozen case studies from the research and computer security communities, going back many years. We provide an overview of various ethical codes, analysis methods, and related discussions from the information warfare and software engineering disciplines.

At the most basic level are issues of privacy that apply across a large percentage of computer security research. It is important that these fundamental issues be addressed, as they are increasingly raised in the context of academic research. Even if a research exception were to be added to the Wiretap Act or Stored Communications Act, the public would likely still want requirements for researchers to adhere to ethical principles that include the kind of minimization techniques described above. Having a legal exception allowing collection of data involving private communications does not mean privacy rights can then be ignored.

At the far end of the spectrum, where the subjects of research are criminal activities that have financial, political, business continuity, or national security implications, there is a need to look beyond privacy rights and harmonize research activities with law enforcement investigation and security- or network-operational requirements. In this area, we need standards and decision-making guidelines that allow deconfliction[1] of researchers' activities, consider alternative actions in terms of risk/benefit, harmonize security operations and research with law enforcement investigations, and balance roles and responsibilities.

We, as a community, urgently need to continue and expand the discussions about sophisticated and potentially aggressive countermeasures to cybercriminal activities in order to minimize harm and maximize benefit in the ongoing conflict occurring in cyberspace.

1. The term *deconfliction* comes from the military, where flight plans of fighters are coordinated to avoid interference during action. In this context, it means coordinating researchers' activities to avoid interfering with each other or interfering with law enforcement investigations, both of which can have negative effects such as over-counting, obscuring criminal actors, or sending law enforcement down dead-end paths that waste scarce resources and time.

REFERENCES

[1] Agora workshop moderators, First Agora Workshop on Active Defense, August 2001: http://staff.washington.edu/dittrich/arc/AGORA\%208JUN01.ppt.

[2] David Dittrich, Second Agora Workshop on Active Defense, September 2003, sponsored by Cisco Systems, Inc.: http://staff.washington.edu/dittrich/arc/AD-workshop-091203.pdf.

[3] David Dittrich, Michael Bailey, and Sven Dietrich, "Have We Crossed the Line? The Growing Ethical Debate in Modern Computer Security Research," November 2009. Poster presented at the 16th ACM Conference on Computer and Communication Security: http://www.sigsac.org/ccs/CCS2009/pd/abstract_22.pdf.

[4] David Dittrich, Michael Bailey, and Sven Dietrich, "Towards Community Standards for Ethical Behavior in Computer Security Research," Technical Report CS 2009-01 (April 20, 2009), Stevens Institute of Technology: http://staff.washington.edu/dittrich/papers/dbd2009tr1/.

[5] David Dittrich and Kenneth E. Himma, "Active Response to Computer Intrusions," Chapter 182 in Vol. III, *Handbook of Information Security* (Wiley, 2005): http://papers.ssrn.com/sol3/papers.cfm?abstract_id=790585.

[6] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher, *Internet Denial of Service: Attack and Defense Mechanisms* (Prentice Hall PTR, 2004).

[7] Stevan D. Mitchell and Elizabeth A. Banker, "Private Intrusion Response," 1998: http://jolt.law.harvard.edu/articles/pdf/v11/11HarvJLTech699.pdf.

[8] President's Commission on Critical Infrastructure Protection, PCCIP –reports archive: http://cip.gmu.edu/clib/PCCIPReports.php.

[9] Several, Results of the Distributed-Systems Intruder Tools Workshop, CERT/CC, December 1999: http://www.cert.org/reports/dsit_workshop.pdf.

JEREMIAH GROSSMAN

top 10 Web hacking techniques: "what's possible, not probable"

Jeremiah Grossman, founder and CTO of WhiteHat Security, is a world-renowned Web security expert. A co-founder of the Web Application Security Consortium (WASC), he was named to *InfoWorld*'s Top 25 CTOs in 2007 and is often quoted in major publications such as *SC Magazine*, *Forbes*, and *USA Today*. He has authored dozens of articles and white papers, is credited with the discovery of many cutting-edge attack and defensive techniques, and is a co-author of *XSS Attacks: Cross Site Scripting Exploits and Defense*. Grossman is also an influential blogger who offers insight and encourages open dialogue regarding research and vulnerability trends. Prior to WhiteHat, Grossman was an information security officer at Yahoo!, responsible for performing security reviews on the company's hundreds of Web sites.

jeremiah@whitehatsec.com

NEW ATTACK TECHNIQUES PROVIDE keen insights into the state of the security of the Web. Web client attack techniques impact online businesses, reveal what may become the next popular exploit technique, and may affect anyone who uses a Web browser. In this article, I cover the top 10 Web hacking techniques, a selection chosen by a panel of security experts from a field of 70 candidates [1].

Sharing technical details of these hacking techniques isn't meant to give malicious hackers a set of instructions, but to level the playing field for the good guys. Without this information, defenders would be unfairly handicapped against a determined criminal element who targets the Web as their primary attack vector. Notification of vendors by researchers also provides vendors with a chance to patch their software before it can be exploited.

It is unclear which of these, if any, will become a widely used method of attack. What we do know is that some have already been used against us. The following hacking techniques were ranked by a panel of four widely recognized security experts (Rich Mogull, Chris Hoff, H.D. Moore, and Jeff Forristal) based on their novelty, impact, and pervasiveness. With that I give you the Top 10 Web Hacking Techniques of 2008!

1. GIFAR by Billy Rios, Nathan McFeters, Rob Carter, and John Heasman [2]

A GIFAR is the concatenation of a GIF image and Java Archive (JAR) containing a potentially malicious Java Applet. Many Web sites take ownership of user-supplied content (e.g., image uploads) after parsing the bytestream beginning to end and ignoring trailing "garbage" data. In the case of GIFAR the trailing garbage data is a compressed Java Applet, a JAR, which is essentially a zip file parsed bottom up. When Web sites take ownership of a GIFAR because it "looks" like a GIF, the attached Java archive may execute arbitrary applet code in the victim's browser under the context of the domain from where it was loaded. This results in a same-origin policy violation, similar in scope to that of a persistent cross-scripting vulnerability. Furthermore, the GIF portion of GIFAR can be substituted for any file type the Web site will accept and parse in a top-down fashion (i.e., JPG, DOC, MP3, etc.).

2. Breaking Google Gears' Cross-Origin Communication Model by Yair Amit [3]

Google Gears is a browser extension that allows developers to create rich and responsive Web applications. Of the many available features, Google Gears offers developers cross-origin communication capabilities, making it much easier to implement mash-ups, for example. Under some circumstances the cross-origin communication security model of Google Gears may be bypassed by an attack that inserts malicious code. If an attacker can upload arbitrary "worker" code (the JavaScript code that can access Gears capabilities) to target a Web site, the attacker can issue malicious commands under that domain. This worker code is likely to pass input security controls, as it lacks suspicious tokens such as <script> tags.

3. Safari Carpet Bomb by Nitesh Dhanjani [4]

The Safari Carpet Bomb attack allows a malicious Web site to litter the user's desktop on Microsoft Windows or the user's "Downloads" directory on OS X with arbitrary files or malware. Unless patched, when the Safari browser is served a file with a content type that cannot be rendered by the browser, it automatically downloads it to the default download location without notifying or asking the user. This "carpet bomb" attack may trick users into clicking on the malicious files by mistake or through curiosity. Safari Carpet Bomb has the distinction of bringing the term "blended threat" into the security vernacular, because if you are able to litter user's machines with arbitrary files, you can further the impact and affect other applications that trust content on the local file system.

4. Clickjacking/Videojacking by Jeremiah Grossman and Robert Hansen [5]

Think of any button (image, link, form, etc.) on any Web site that can appear between the Web browser walls. This includes wire transfer forms from bank sites, DSL router buttons, Digg buttons, CPC advertising banners, Netflix queue, Facebook friend requests, and so on. Next consider that an attacker can invisibly hover these buttons below the user's mouse using iframe tags and CSS opacity functionality, so that when a user clicks on something they visually see, they're actually clicking on something the attacker wants them to—you now have clickjacking. We also demonstrated that clickjacking can be used to trick users into enabling a Web cam and microphone through a Flash movie to enable remote surveillance. If you haven't done so already, I strongly suggest you upgrade to Flash version 10 or later or at least cover up the camera with a Post-It note. Finally, cross-site request forgery defenses using one-time tokens (nonces) can also be bypassed using clickjacking.

5. A Different Opera by Stefano Di Paola [6]

Until it was patched, the Opera Web browser itself was vulnerable to a cross-site scripting vulnerability in the History Search page, where JavaScript execution occurred under the opera:* context. Using iframe tags and a cross-site request forgery, this provided a malicious attacker with the ability to modify browser settings under opera:config, specifically the "mailto" preference. Updating the mailto preference to an arbitrary value could enable the arbitrary execution of operating system commands.

6. Abusing HTML 5 Structured Client-Side Storage by Alberto Trivero [7]

HTML 5 has introduced three powerful new ways to store significant amounts of data on the client's PC through the browser. This allows storage of much more data than standard cookies, in Session Storage, Local Storage, and Database Storage. If a Web application using this kind of client-side storage is vulnerable to cross-site scripting, attackers can use their payload to read or modify the content of known storage keys on the computer's victim. If the Web application loads data or code from the local storage, this could also be a powerful method to inject malicious code that will be executed every time the Web application requests it.

7. Cross-Domain Leaks of Site Logins via Authenticated CSS by Chris Evans and Michal Zalewski [8]

Web browser vendors take great pains to ensure that their same-origin policy prevents code on one Web site from obtaining details, such as authenticated content or session cookie data, from another Web site. Violations of the same-origin policy, such as the ability to determine if a user is actively logged on to an arbitrary Web site (e.g., a social network), has serious security and privacy implications. One way this can be achieved is through the inline inclusion of authenticated Cascading Style Sheets on off-domain locations by a malicious Web page. The malicious Web page checks to see if unique CSS properties have been loaded by the off-domain Web page using standard JavaScript APIs. If so, the user is logged in—a simple Boolean result. Similarly, this same attack can be performed with content that only appears in authenticated sessions, including images and JavaScript files.

8. Tunneling TCP over HTTP over SQL Injection by Glenn Wilkinson, Marco Slaviero, and Haroon Meer [9]

The common Web infrastructure is designed using a multi-tier architecture. A client connects to the server (port 80/443), which connects to back-end databases and applications to generate dynamic content. Remote clients may not directly connect to the back-end systems, where the crown jewels are located, as the server can, and certainly cannot communicate with them over arbitrary protocols and ports—that is, unless the server has a SQL injection vulnerability. In this technique, squeeza, a tool for exploiting SQL injection, is used to upload reDuh to the vulnerable server as a JSP, PHP, or ASP file. reDuh, when executed as a Web application on the vulnerable server, creates a TCP tunnel through validly formed HTTP requests using a client-server model. reDuh gives an attacker access to the server behind the first-layer firewall, which then acts as a relay to communicate with any reachable back-end system.

9. ActiveX Repurposing by Haroon Meer [10]

Resident or latent ActiveX controls, including those used to access SSL VPNs, can be abused by a malicious attacker. In this technique, a particular ActiveX control included the features to update itself if the server informed it of a new software version. By simply instantiating the control and passing it a higher build number and a URL path to a downloadable file, it would cause the client to download a possibly malicious file. Before loading the control, Internet Explorer would first check the downloaded file to see if it was properly signed. If it was not, then the file would not be executed. However, the file would still download to a predictable location on the local file system, where it would remain. Upon first malicious instantiation, an attacker would force the control to download a mock configuration file it supported. The second instantiation would call the control and point to the previously downloaded configuration file, which could contain arbitrary operating system commands, including an uninstall method.

10. Flash Parameter Injection by Yuval Baror, Ayal Yogev, and Adi Sharaban [11]

Flash parameter injection introduces a new way to inject values into global parameters in Flash movies while the movie is embedded in its original HTML environment. These injected parameters can grant the attacker full control over the page DOM, as well as control over other objects within the Flash movie. This can lead to more elaborate attacks which take advantage of the interaction between the Flash movie and the HTML page in which it is embedded. There are several different FPI variants, and most include tricking the server into sending back a page where user input is interpreted as Flash parameters. This allows an attacker to inject malicious global parameters to the Flash movie and exploit Flash-specific vulnerabilities. When an attacker is able to access and control global Flash parameters, he can achieve attacks such as cross-site scripting through Flash, cross-site flashing, and changing the flow of the Flash video.

Conclusion

There is a difference between what is possible and what is probable, something we often lose sight of in the world of information security. For example, a vulnerability represents a weakness an intruder may exploit in an asset by way of a particular attack technique, such as those described above. Obviously, a vulnerability's mere existence does not necessarily mean it will be exploited or indicate by whom or to what extent. Some vulnerabilities are more difficult to exploit than others and therefore attract different attackers. Often a particular attack technique will only become widely used maliciously years after initial discovery, similarly to SQL injection. This is why we are exploring them now.

What we do know is that attack techniques tend to only be taken seriously after they are both well understood and respected. We can assist with understanding through awareness efforts but, unfortunately, historically respect is gained through mass exploitation.

REFERENCES

[1] Top Ten Techniques blog entry: http://jeremiahgrossman.blogspot.com/2009/02/top-ten-web-hacking-techniques-of-2008.html.

[2] GIFAR technique: http://xs-sniper.com/blog/2008/12/17/sun-fixes-gifars/.

[3] Breaking Google Gears' cross-origin communication protection: http://blog.watchfire.com/wfblog/2008/12/breaking-google-gears-cross-origin-communication-model.html.

[4] Safari Carpet Bombing: http://www.dhanjani.com/blog/2008/05/safari-carpet-b.html.

[5] Clickjacking: http://www.sectheory.com/clickjacking.htm.

[6] Opera History attack: http://seclists.org/fulldisclosure/2008/Oct/0401.html.

[7] Abusing HTML 5 Structured Client-Side Storage: http://trivero.secdiscover.com/html5whitepaper.pdf.

[8] Cross-domain leaks of site logins via Authenticated CSS: http://scarybeastsecurity.blogspot.com/2008/08/cross-domain-leaks-of-site-logins.html.

[9] Tunneling TCP over HTTP over SQL Injection: http://www.sensepost.com/research/reDuh/.

[10] ActiveX Repurposing: http://www.sensepost.com/blog/2237.html.

[11] Flash Parameter Injection: http://blog.watchfire.com/wfblog/2008/10/flash-parameter.html.

GIORGIO MAONE

hardening the Web with NoScript

Giorgio Maone is CEO and CTO of InformAction, a software development and IT consulting firm based in Italy. He's the author and main developer of NoScript, a popular open source solution enhancing browser security.

g.maone@informaction.com

NOSCRIPT IS A POPULAR SECURITY add-on for Firefox and other Web browsers based on Mozilla technology. Although it is mainly known for providing easy fine-grained script blocking at the domain level, NoScript pioneered several innovative and unique client-side countermeasures against emergent Web-based threats, such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and UI redressing (also known as "clickjacking"), which had previously believed to be addressable on the server side only.

Default Deny, Easy Allow

Since its very first release (May 2005 [1]), NoScript's core feature has been whitelist-based selective activation of "executable" Web content. JavaScript and browser plugins containing scripting interpreters and just-in-time compilers, such as Sun's Java, Adobe's Flash, or Microsoft's Silverlight, have turned the Web, which had been originally intended as an interlinked collection of static documents, into a rather anarchic executable platform with loose or nonexistent security checks. The *same-origin policy* (stating that active Web content on a certain site must not be allowed to access data or execute code from a different site) and sandboxing mechanisms (meant to prevent active Web content from reaching out of the browser and interacting with the underlying system) have long been the only security constraints enforced by browsers on Web-based "programs," but they get violated very often, because of design flaws or implementation bugs.

Design flaws, responsible for "structural" vulnerabilities such as XSS, CSRF, or UI redressing, are very unlikely to be fixed in a satisfactory way, because of compatibility concerns: their mitigation is delegated to Web development "good practices" recommendations, doomed to remain almost always unheard or misunderstood. Implementation bugs, often allowing malicious Web content to escalate privileges and compromise a user's account or system, are the reason why Web browsers and their plugins are bound to impressively tight patching and updating cycles, which are indispensable to keep an acceptable degree of security [2]. Unfortunately, the rise of a florid zero-day vulnerabilities black market, full disclosure stunts, corporate rules slowing down or banning automatic updates, legacy compatibility needs, and other factors can significantly widen the exposure window of many users to unpatched browser and plugin vulnerabilities, which have quickly become a major venue of malware propagation.

Nearly every security vulnerability that has been affecting the browser or its plugins so far could be mitigated or even, more often than not, completely neutered by disabling JavaScript or, when applicable, the vulnerable plugin. In fact, almost all the security advisories about exploitable browser flaws play the "Disable JavaScript" card as the only possible workaround until a patch is available. However, in the modern Web, where many sites and applications rely on JavaScript-based techniques (e.g., DHTML and AJAX) to enhance users' "experience" or even to implement their basic functionality, this simple and effective countermeasure is often impractical.

But what if you had a quick and easy way to enable JavaScript and potentially dangerous plugins *only on those sites you trust*, either permanently or just when you need to? This is exactly what NoScript has been conceived for: enforcing a "Default Deny" policy on *active* Web content, yet providing users with the ability to whitelist trusted domains on the fly as needed, by popping up a contextual menu and selecting the proper "Allow some.trusted.domain.com" command. A subtle non-modal notification bar is displayed on the bottom of pages where active content has been disabled, to remind you that some script *might* need to be allowed if the site doesn't work properly. This feedback system has been carefully designed to be as discreet as possible and never get in your way, especially on those Web sites which do work fine even if scripting is disabled. NoScript neither begs for attention nor requires any user interaction: it tries to avoid the trap of training users to permit everything, an effect that modal security questions ("Allow this?" "Block that?") are often accused of causing.

Site-level permissions for active Web content actually had a venerable precursor in Microsoft Internet Explorer 6's "Security Zones" [3], and about nine months after NoScript's appearance, Opera 9 provided a user interface for configuring "Site specific preferences" [4], including JavaScript, Java, and Plugins. However, IE's Zones, being mainly oriented to enforcing corporate policies, are buried deep inside the Internet Options panel and quite hard to configure for end users, while Opera's implementation, albeit user-friendlier, lacks any contextual feedback system and the ability to discriminate among third-party imported scripts, both required for effective security-grade script management.

Deflecting Reflective XSS

Assuming that the whitelist policy for active content execution is effectively enforced and cannot be violated—NoScript's implementation has never been broken so far—is there still any way for malicious code to run against a user's will? Sadly, the answer is yes: quite obviously, it is sufficient for the malicious code to be injected in any of the whitelisted sites. This can be achieved by hacking the Web server that hosts the site or, much more frequently, through a Cross-Site Scripting (XSS) attack.

XSS vulnerabilities affect those Web applications which don't properly escape their input when it is echoed back as (X)HTML output: this allows script fragments crafted by the possibly malicious user controlling the input to be executed by the browser in the context of the vulnerable site. According to studies by the Web Application Security Consortium [5] and WhiteHat Security [6], corroborated by live data from the XSS Project [7], this kind of vulnerability is the most prevalent in Web application security and affects an overwhelming majority of Web sites, from social networks to online banking applications, no matter how popular and/or resourceful they are. Of course, XSS lowering the effectiveness of script blocking is a minor concern compared to its overall impact: XSS attacks can be used to silently steal credentials, perform stealth financial transactions impersonating a logged-in user, or set up "perfect" phishing attacks (undetectable, since the fake page comes from the real domain).

The painful awareness of these threats and the complete lack of initiative by the browser vendor against them, being considered at the time a server-side only problem which could never be mitigated on the client side, ignited the development of the first in-browser XSS filter, which was publicly released as a NoScript component called InjectionChecker in March 2007. InjectionChecker examines any cross-site HTTP request for HTML documents, looking for HTML or JavaScript fragments that could be injected in the destination page. If one is found, the request gets sanitized by stripping out the potential payload before it's sent. Initially considered with skepticism by both security researchers and browser vendors, this approach quickly demonstrated its reliability and effectiveness against Type 0 (DOM-based) and Type 1 (Reflective) XSS attacks. The main limitation of the earliest InjectionChecker versions, which were based exclusively on pattern matching, was a moderately high false-positive rate. However, after some development iterations, the analysis algorithms underwent a radical overhaul: by leveraging the browser's JavaScript interpreter itself (SpiderMonkey) in order to discriminate non-trivial and syntactically valid script injections from innocuous but suspect request data, newer versions managed to reduce the false-positive rate near to 0. Still, even though extremely rare, a cross-site request might legitimately include some valid HTML or JavaScript fragment and therefore trigger the InjectionChecker. However, this residual issue is alleviated by the non-blocking design of the filter which, rather than preventing the possibly attacked page from loading or brutally disabling its scripting capabilities, just sanitizes the request, modifying the bare minimum for the attack to fail: this approach usually keeps the landing page functional. Furthermore, the issued warning message is non-modal (like every notification from NoScript) and gives the user an option to examine the original request and replay it unfiltered, if it is deemed safe. Finally, exceptions for safe origins or targets can easily be configured to handle specific situations.

The success of NoScript's XSS filters probably encouraged browser vendors to approach this problem with fewer prejudices. In fact, even if more than one year later, Microsoft revealed that an XSS protection subsystem, impressively resembling NoScript's InjectionChecker, was being added to Internet Explorer 8 [8], and in September 2009 Adam Barth announced a similar development effort in progress for the open source Chromium browser on which Google Chrome is based [9]. Notwithstanding, both Microsoft's and Google's solutions appear quite limited compared to NoScript's: since they act on the page rather than on the request, they're unable to neutralize Type 0 (DOM-based) XSS and, at least in Microsoft's case, new XSS vulnerabilities can be introduced by the neutering routine itself, when it modifies the landing document's contents.

ClearClick vs Clickjacking

In September 2008 Jeremiah Grossman (WhiteHat Security) and Robert "RSnake" Hansen (SecTheory) generated lots of buzz when, requested by Adobe, they canceled a speech scheduled for the World OWASP AppSec conference in New York. A new exploitation technique they were going to present, dubbed “clickjacking,” implied many more critical consequences than initially thought, if combined with an otherwise minor flaw in the Flash browser plugin [10]. As was revealed after Adobe had fixed its plugin-specific issue, a remote attacker could easily modify the local Flash privacy settings and start spying on a user’s activity through his microphone or Webcam [11].

While speculations about the nature of this mysterious attack flourished, some observers deduced from the available information that, even if the specific exploitation scenario was indeed new and spectacular, the underlying vulnerability was a known one, endemic in all the modern browsers but still underestimated (or, better, understated, because no obvious solution could be deployed without drastically breaking the Web as we know it): UI redressing [12]. This is the problem definition as put by Google’s browser security expert Michal Zalewski:

> A malicious page in domain A may create an IFRAME pointing to an application in domain B, to which the user is currently authenticated with cookies. The top-level page may then cover portions of the IFRAME with other visual elements to seamlessly hide everything but a single UI button in domain B, such as “delete all items,” “click to add Bob as a friend,” etc. It may then provide [its] own, misleading UI that implies that the button serves a different purpose and is a part of site A, inviting the user to click it. Although the examples above are naive, this is clearly a problem for a good number of modern, complex Web applications. [13]

UI redress/clickjacking, in its simplicity, is actually much more faceted and difficult to approach than it seems: variants may target same-site plugin content (as in the famous Adobe case) rather than cross-site documents, the victim UI can be rendered transparent by abusing the CSS “opacity” property rather than by being covered by the parent malicious site, keyboard strokes might be solicited rather than clicks, and so on. In spite of the fact that NoScript, as noted by Jeremiah Grossman in his early interviews before full disclosure, provided protection against his Flash-based clickjacking exploit by default and against the more general scriptless UI redress attacks if users enabled the “Forbid IFrames” option, the latter configuration was much too inconvenient to be recommended to the general public.

There was clearly a need for a specific countermeasure, which had not been conceived yet. So on October 7, 2008, after a week-long design and coding marathon, a prototype of the ClearClick NoScript module could be finally released [14]. ClearClick’s concept is almost as simple as UI redressing itself: whenever a mouse or keyboard interaction is engaged with a cross-site framed document or an embedded plugin object, event processing gets temporarily suspended while two screenshots of the involved item are compared: one taken from the top-level window (reproducing the user’s point of view), the other taken after isolating and opacizing the event target. If the two images match, the user can see “the naked truth” and the original mouse or keyboard event processing is immediately resumed. Otherwise, the situation is considered suspect because the event target is concealed, transparent, or otherwise not clearly visible: a warning is issued, showing both the screenshots for easy visual verification and allowing the user to judge if the interaction needs to be aborted or not.

Some months later Microsoft announced with a fanfare [15] that “clickjacking protection” was being added to IE8, but it was quickly exposed [16] as an “X-Frame-Options” HTTP header which should be sent by Web sites when they do not want to be framed: an opt-in proposal requiring Web developers’ cooperation, then, not comparable to a client-side automatic solution like ClearClick. Nevertheless, NoScript implemented this feature as well (just a few hours after it had been revealed) for compatibility's sake, while Apple's Safari 4 and Google's Chrome 2 followed the lead later. However, as Google's "Browser Security Handbook" itself explains,

> So far, the only freely available product that offers a reasonable degree of protection against the possibility is NoScript (with the recently introduced ClearClick extension). To a much lesser extent, an opt-in defense is available [for] Microsoft Internet Explorer 8, Safari 4, and Chrome 2, through a X-Frame-Options header, enabling pages to refuse being rendered in any frames at all (DENY), or in non-same-origin ones only (SAMEORIGIN) [18].

ABE Patrolling the Web's Borders

Cross-Site Request Forgery (CSRF) had been called "the sleeping giant" [19] back in 2006, because it was as ubiquitous as it was misunderstood. If a Web application is vulnerable, a malicious site can perform unintended actions (e.g., to transfer funds or change router settings) on behalf of the users who are browsing it, by silently sending a known HTTP "command" request through one of the many automatic navigation vehicles provided by HTML and JavaScript. The browser will automatically add authorization information, either as a session cookie or an Authorization header. Three years later the giant has awakened, even though some progress has been done in prevention: awareness grew among developers, and support for countermeasures, such as explicit security tokens, has been introduced in popular Web application frameworks.

However, as usual, mitigation is left to Web authors' skill and good will, with no help from the client side and no control in user's hands. ABE (Application Boundaries Enforcer), a project sponsored by the NLnet Foundation [20], tries to improve this situation.

Released as a NoScript component in June 2009 [21], but planned to be also decoupled from the Firefox add-on and ported to different browsers, ABE is a firewall-like system which allows users, Web developers or trusted third parties (subscription providers) to configure "Rulesets" declaring the boundaries of one or more Web applications. Rules are expressed using a syntax [22] which should look natural to any system administrator. This rule, for instance, can be used to protect Gmail against CSRF attacks:

```
Site mail.google.com
Accept from SELF, www.google.com
Deny
```

It causes Gmail (mail.google.com) to reject (`Deny`) all the potentially forged requests, identified as those coming from any site except mail.google.com itself (`SELF`) and www.google.com, the domain from which the login form for the Google application is served. Selectors can be much more fine-grained than these, allowing glob patterns and regular expressions to be combined in site specifications and HTTP methods to be used as criteria to match requests. Documentation and examples are available on the project Web site, http://noscript.net/abe/.

Rules are enforced at the beginning of the HTTP load cycle, preventing malicious requests from doing any harm. Furthermore, since it lives inside the browser, ABE knows the real origin of each request, allowing decisions to reliably depend on this information but not requiring it to be leaked through the wire, unlike the Referer HTTP header which, indeed, often gets suppressed or forged because of privacy concerns and, for this reason, must not be trusted.

Any Web site can protect its boundaries by providing an ABE rule set in its root directory, but they can't override the user's own rule sets or those on other sites. ABE refreshes site-provided rule sets when a session starts, then hourly, but honors HTTP caching hints if provided.

Users can add their own rules, which take precedence over the ones pushed by trusted third parties and Web applications, by editing the initially empty USER rule set accessible from the ABE panel, among NoScript's Advanced options. A visual UI to build rules contextually, during navigation, is under development.

The only ruleset provided at installation time, labeled SYSTEM, includes just one rule:

```
Site LOCAL
Accept from LOCAL
Deny
```

This quite obviously means that requests toward local sites (i.e., private IPv4 and IPv6 networks according to RFC 3330 and RFC 4193) are blocked unless they come from origins which are local as well. Such a rule automatically protects intranets against scanning and CSRF attacks toward internal Web applications and devices (e.g., router hacking) initiated from malicious Internet Web sites [23].

Did You Know?

Although often described as a "simple" script blocker, NoScript features multiple additional security enhancements, completely independent of its script-blocking core. Some users may believe that maintaining a whitelist of trusted sites allowed to run scripts is too tedious in this AJAXified world. Nevertheless, they should give NoScript a try: no matter if they give up and resort to "Allow Scripts Globally (dangerous!)," the InjectionChecker, ClearClick, and ABE components, unattended and silent in the background, will keep delivering a degree of protection against XSS, clickjacking, and CSRF that is currently unmatched by any other available Web browser technology.

REFERENCES

[1] NoScript's public release versions history: https://addons.mozilla.org/en-US/firefox/addons/versions/722.

[2] T. Duebendorfer and S. Frei, "Why Silent Updates Boost Security," *ETH Tech Report 302,* May 5, 2009: http://www.techzoom.net/publications/silent-updates/.

[3] http://www.microsoft.com/windows/ie/ie6/using/howto/security/setup.mspx.

[4] "Opera 9 introduces 'Site specific preferences' User Interface," February 7, 2006: http://snapshot.opera.com/windows/w90p2.html

[5] Web Application Security Consortium, "Web Application Security Statistics 2007": http://www.Webappsec.org/projects/statistics/.

[6] WhiteHat Website Security Statistics Report: http://www.whitehatsec.com/home/resource/stats.html.

[7] The XSS Project: http://www.xssed.com.

[8] Giorgio Maone, "NoScript's Anti-XSS Filters Partially Ported to IE8," July 3, 2008: http://hackademix.net/2008/07/03/noscripts-anti-xss-filters-partially-ported-to-ie8/.

[9] Adam Barth, "Reflective XSS protection (for reals this time)," Chromium-dev Group, September 4, 2009: http://groups.google.com/group/chromium-dev/browse_thread/thread/d2931d7b670a1722/d56bdfccfcef677f.

[10] Robert Hansen, "Clickjacking," September 15, 2008: http://ha.ckers.org/blog/20080915/clickjacking/.

[11] Robert Hansen and Jeremiah Grossman, "Clickjacking," September 12, 2008: http://www.sectheory.com/clickjacking.htm.

[12] Mark Pilgrim, "This Week in HTML 5—Episode 7," September 29, 2009: http://blog.whatwg.org/this-week-in-html-5-episode-7.

[13] Michal Zalewski, "Dealing with UI Redress Vulnerabilities Inherent to the Current Web," WHATWG Mailing List, September 25, 2009: http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2008-September/016284.html.

[14] Giorgio Maone, "Hello ClearClick, Goodbye Clickjacking": http://hackademix.net/2008/10/08/hello-clearclick-goodbye-clickjacking/.

[15] Giorgio Maone, "Ehy IE8, I Can Has Some Clickjacking Protection?" January 27, 2009: http://hackademix.net/2009/01/27/ehy-ie8-i-can-has-some-clickjacking-protection/.

[16] Giorgio Maone, "IE8's 'Clickjacking Protection' Exposed," January 28, 2009: http://hackademix.net/2009/01/28/ie8s-clickjacking-protection-exposed/.

[17] Giorgio Maone, "X-FRAME-OPTIONS in Firefox," January 29, 2009: http://hackademix.net/2009/01/29/x-frame-options-in-firefox/.

[18] Michal Zalewski (Google Inc.), "Arbitrary Page Mashups (UI Redressing)," Browser Security Handbook: http://code.google.com/p/browsersec/wiki/Part2#Arbitrary_page_mashups_%28UI_redressing%29.

[19] Jeremiah Grossman, "CSRF, the Sleeping Giant," September 26, 2006: http://jeremiahgrossman.blogspot.com/2006/09/csrf-sleeping-giant.html.

[20] NLnet Foundation's NoScript/ABE page: http://www.nlnet.nl/project/noscriptabe/.

[21] Giorgio Maone, "Meet ABE," June 30, 2009: http://hackademix.net/2009/06/30/meet-abe/.

[22] Giorgio Maone "ABE—Rules Syntax and Capabilities": http://noscript.net/abe/abe_rules.pdf.

[23] Jeremiah Grossman, "Hacking Intranet Websites from the Outside," Black Hat (USA)—Las Vegas, August 3, 2006: http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf.

GERWIN KLEIN

correct OS kernel? proof? done!

Gerwin Klein is a principal researcher at NICTA in Sydney, Australia. He is leading the L4.verified project at NICTA and teaches formal program verification at the University of New South Wales.

gerwin.klein@nicta.com.au

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and by the Australian Research Council through the ICT Centre of Excellence program.

TWO YEARS AGO GERNOT HEISER demanded in this venue *Your System is Secure? Prove it!* [5] He also mentioned the L4.verified [3] project at NICTA, which is doing just that. This proof is now completed, and in this article I show what we have proved and what that means for security.

The seL4 Microkernel: Correct!

The basic idea goes back to the 1970s: off and on since then, people have been trying to formally verify operating systems [10, 4]. It's the obvious place to start when you are serious about meaningful assurance for critical systems. The idea for formal verification is that programs are just mathematics in the end, and if you want to show beyond doubt that something is true in mathematics, you prove it. If you want to be sure that the proof is right, you do it fully formally so that it can be machine-checked.

It was clear early on that this is possible in principle, but enthusiasm ebbed after an initial flurry of activity around the late '70s and early '80s. Mathematical semantics for real programming languages had not developed far enough, machine support for theorem proving was only starting to appear, and the whole problem seemed infeasible for any real program of interesting size. Full formal program verification was like controlled fusion power: about 30 years of research in the future.

In contrast to controlled fusion, 30 years later things have changed. With the formal verification of the seL4 microkernel, we have reached an important milestone: the first commercially viable microkernel, formally verified all the way down to its low-level C implementation [8]. The proof is machine-checked from first principles in the theorem prover Isabelle/HOL [2], and it was an order of magnitude cheaper to build than a traditional software certification.

seL4 is a small microkernel in the L4 family [1]: 8,700 lines of C and 600 lines of assembly. It is not Linux with millions of lines of code. Instead, it provides the basic mechanisms to build an OS: threads, message passing, interrupts, virtual memory, and strong access control with capabilities. Network, file systems, and device drivers are implemented in user space in microkernel systems, and it has been shown that this can be achieved with high performance.

Our proof does precisely what the dream of the '70s was: we define formally in an abstract specification what it means for the kernel to be correct. We describe what it does for each input (trap instruction, interrupt, etc.) but not necessarily how it is done. Then we prove mathematically that the C implementation always correctly implements this specification.

The proof goes down to the level of the C implementation and assumes the correctness of things below that level: compiler, linker, assembly code, hardware, low-level cache management, and TLB. We also assume correctness of the boot code. This is still a long list, but each proof has to stop somewhere. This is what we picked. With more resources, it is possible to eliminate almost all of the assumptions above. There are, for instance, a number of recent research projects on verified optimizing C compilers.

We not only proved the code correct, we also did extensive design proofs and validation.

What Does This Mean for Security?

In a sense, functional correctness is one of the strongest properties you can prove about a system: you have a precise formal prediction of how the kernel behaves in all possible situations for all possible inputs. If you are interested in a more specific property and you can express this property in Hoare logic, it is now enough to work with this formal prediction, with the specification. This is orders of magnitude easier than direct proofs on the implementation.

Does this mean that we have proved seL4 is secure? Not yet, really. We proved that seL4 is functionally correct. *Secure* would first need a formal definition, which in itself is a wide field and depends on what you want to use the kernel for. Taken seriously, security is a whole-system question, including the system's human components. Nevertheless, there are many different formal security properties of the kernel, such as access control, confidentiality, integrity, and availability that one might be interested in. We do have a high-level model for seL4 access control with a nice confinement theorem about it, and we are currently busy connecting up this model with the proven specification. Other properties, for instance secrecy, do not necessarily connect that easily, but we will be looking into that as well in the future.

Even without proving specific security properties on top, a functional correctness proof already has interesting implications for security. If you sit back a minute and think about what it means that we can always predict what the system is going to do, then a number of things come to mind. If the assumptions above are true, in seL4 we will have:

- **No code injection attacks.** If we always know precisely what the system does, and if the spec doesn't explicitly allow it, then we can't have any foreign code executing as part of seL4. Ever.
- **No buffer overflows.** This is mainly a classic vector for code injection, but buffer overflows may also inject unwanted data and influence kernel behavior that way. We prove that all array accesses are within bounds, and we prove that all pointer accesses are well typed, even if they go via casts to void and arcane address arithmetic. Buffer overflows are not going to happen.
- **No NULL pointer access.** Few things crash a kernel more nicely or are easier to exploit: see, for instance, a recent bug in the Linux kernel believed to affect all versions since May 2001 [9]. The bug allows local privilege escalation and execution of arbitrary code in kernel mode, and it's a classic

NULL pointer dereference. We have these as direct proof obligation for every pointer access in the system. None of them occurs in seL4.

- **No ill-typed pointer access.** Even though the kernel code deliberately breaks C type safety for efficiency at some points, in order to predict that the system behaves according to specification, we have to prove that circumventing the type system is safe at all these points. We cannot get unsafe execution from any of these operations.
- **No memory leaks.** There are no memory leaks in seL4 and there is never any memory freed that is still in use. This is not purely a consequence of the proof itself. Much of the design of seL4 was focused on explicit memory management, and it is one of the kernel's more innovative features. Users may run out of memory, but the kernel never will. The kernel also provides an availability property for users (this one we have not yet explicitly proved): if you have set up your memory resources correctly, other users will not be able to starve you of that memory or of the kernel memory resources necessary to back your metadata. This is far from true for other kernels, even in the L4 family.
- **No non-termination.** We have proved that all kernel calls terminate. This means the kernel will never suddenly freeze and not return from a system call. This does not mean that the whole system will never freeze; you can still write bad device drivers and bad applications, but if you set it up right, you can always stay in control of runaway processes.
- **No arithmetic or other exceptions.** The C standard defines a long list of things that can go wrong and that you should not be doing: shifting machine words by too large an amount, dividing by zero, etc. Our framework makes it a specific obligation for us to prove that these do not occur. We have solved all of them. We're also taking special care with overflowing integer arithmetic.
- **No unchecked user arguments.** All user input is checked and validated. If the kernel receives garbage or malicious packages it will respond with the specified error messages, not with crashes. Of course, it is still possible to shoot yourself in the foot. For instance, if you have enough privileges, the kernel happily allows a thread to kill itself. It will never allow anything to crash the kernel, though.

Many of these are general security traits that are good to have for any kind of system. We have also proved a large number of properties that are specific to this kernel. We have proved them about the kernel design and specification. With functional correctness, we know they are true about the code as well. Some examples are:

- **Aligned objects.** Two simple low-level invariants of the kernel are: all objects are aligned to their size, and no two objects overlap in memory. This makes comparing memory regions for objects very simple and efficient.
- **Well-formed data structures.** Lists, doubly linked, singly linked, with and without additional information, are a pet topic of formal verification. These data structures also occur in seL4, and we proved the usual properties: lists are not circular when they shouldn't be; back pointers point to the right nodes; insertion, deletion, etc., work as expected and don't introduce any garbage.
- **Algorithmic invariants.** Many optimizations rely on certain properties being always true, so specific checks can be left out or can be replaced by other, more efficient checks. A simple example is that the distinguished idle thread is always in thread state *idle* and therefore can never be blocked or otherwise waiting for I/O. This can be used to remove checks in the code paths that deal with the idle thread. If the state invariant wasn't true, not having explicit cases for other thread states would easily lead to kernel crashes.

- **Correct bookkeeping.** This one was much more interesting and consists of a large collection of individual properties. The seL4 kernel has an explicit user-visible concept of keeping track of memory, who has access to it, who access was delegated to, and what needs to be done if a privileged process wants to revoke access from a whole collection of delegates. It is the central mechanism for re-using memory in seL4. The data structure that backs this concept is correspondingly complex, and its implications reach into almost all aspects of the kernel. For instance, we proved that if a live object exists anywhere in memory, then there exists a node (an explicit capability, actually) in this data structure that covers the object. And if such a capability exists, then it exists in the proper place in the data structure and has the right relationship toward parents, siblings, and descendants within. Also, if an object is live (may be mentioned in other objects anywhere in the system), then the object itself together with that capability must have recorded enough information to reach all objects that refer to it (directly or indirectly). Together with a whole host of further invariants, these properties allow the kernel code to reduce the complex, system-global test, whether a region of memory is mentioned anywhere else in the system, to a quick, local pointer comparison that takes next to no time to execute.

We have proved about 80 such invariants on the low-level design such that they directly transfer to the data structures used in the C program.

The key condition in all this is *if the assumptions above are true*. To attack any of these properties, this is where you would have to look. What the proof really does is take 7,500 lines of C code out of the equation and reduce possible attacks and the human analysis necessary to guard against them to the remaining bits. It is not an absolute guarantee or a silver bullet, but it is definitely a big deal.

What About Type-Safe Languages and Static Analysis?

A frequent question is whether the same couldn't be achieved by full coverage testing, by using a type-safe language, or by static analysis.

It is almost customary in verification papers to bash testing as not sufficient to show the absence of bugs and therefore useless. I'm not going to do that here. Used correctly and in combination with other techniques, testing is an effective method to get reliable software. After all, planes do not fall out of the sky all the time because of implementation errors. There have been software-related incidents, but to the best of my knowledge planes still are the safest mode of transportation available. People *can* build reliable software. Testing is just very hard and very expensive to get complete. There are no easy measures for it. For example, if you have the very simple fragment of C code `if (x < y) z = x/y else z = y/x` for `x`, `y`, and `z` being `int` and you test with `x=4, y=2` and `x=8, y=16`, you have full code coverage, every line is executed at least once, every branch of every condition is taken at least once, and you still have two bugs remaining. Of course, any human tester will immediately spot that you should test for something like `x=0, y=-1` and `x=-1, y=0`, but for bigger, non-trivial programs there is no easy way to find these cases and it is pretty much infeasible to be sure you have all of them. Humans are good at ingenuity and creativity, but they are not so good at repetitive, complex, high-detail tasks—especially not under pressure. So anything that can reduce the burden should be used. Our style of formal verification, on the other hand, is very good at completeness. It's what it's all about. It will force you to work through all the relevant cases, and it will tell you what the relevant cases are. And because humans are bad at repetitive tasks, we have machine assistance and machine-checking of the proofs.

As I said above, the verification takes the C implementation out of the picture. You now only have to test the models and the specification against your expectations. The C model can be reused for any verification, so that cost amortizes quickly. Testing the specification was a big part of our development and design process and is a lot easier than testing the implementation.

Similarly, type-safe languages as used in Singularity [7], for instance, are good. They help. They prevent you from doing lots of stupid things right from the start. If you can, you should use them. But they will usually not save you from the effects of a NULL pointer dereference. Sure, an unexpected NULL pointer access will be checked, caught, and reported as an exception. But in an OS, then what? You may fail gracefully instead of catastrophically, but even that may be hard to do if you have progressed way after argument checking and have already changed parts of the state. The difference is that seL4 will just not access NULL pointers. Period.

Type-safe languages often require a complex runtime of "dirty" code that needs to be trusted. Singularity's trusted code base is larger than the whole seL4 kernel, so there is no real win in terms of easier verification. And despite Singularity eliminating the need for context switches, you still pay overhead for runtime checks. As a result, L4 kernels are still faster [6].

Static analysis can do even more than most type-safe languages. Some parts of the security-relevant properties that are implications and by-products of our proof are covered in theory by a number of static analysis tools. The advantage and the problem with static analysis is that it is automatic. It cannot be safe and complete at the same time, otherwise it would solve the halting problem. So if it is safe, then it will have false alarms and instances of *maybe correct* instead of *definitely correct*. The functional correctness property we proved is way too hard for static analysis. Even the by-products are too hard. Some instances of pointer dereferences in the code are safe for deep reasons of the underlying algorithm. You would have to add redundant explicit checks into the code to make them go through with static analysis. This is precisely what you don't want for a high-performance OS kernel. Humans, constructing an interactive, machine-checked proof, on the other hand, have no problem solving the halting problem and conducting a fully precise analysis.

So, Did You Find Any Bugs?

We didn't test the kernel extensively before verification started, but we did quite a bit of debugging initially and we did use it for student projects internally for more than six months and ported it to a different architecture. After initial debugging, these activities found 16 bugs in this internal alpha-release. After that, the kernel ran just fine for everything the students wanted to do.

We also ran a static analysis tool on the code before verification. We found two bugs (counted in the above 16) and got hundreds of false positives.

Formal verification then found 144 more bugs in the C code, in total 160. This means that even though the code appeared to be running just fine for normal application, there was an order of magnitude more bugs lurking in there than the 16 found initially. They were mostly but not exclusively in rarely used features. As mentioned, we also did proofs on the design and the specification level. Most of these design and specification proofs were completed before the code was written, and we fixed about 150 issues in each. That means, in total we have discovered roughly 460 issues in kernel code, design, and specification.

None of the bugs found in the C verification stage was deep in the sense that the corresponding algorithm was flawed. These were already caught in the design validation phase. Some examples are missing checks on user supplied input or subtle side effects in the middle of an operation breaking global invariants. The bugs discovered in the C code were mainly typos, misreading the specification, or failing to update all relevant code parts for specification changes. Even though their cause was often simple, understandable human error, their effect in almost all cases was sufficient to crash the kernel or create security vulnerabilities. Other more interesting bugs found during the C implementation proof were missing exception case checking and different interpretations of default values in the code. For example, the interrupt controller on ARM returns 0xFF to signal that no interrupt is active, which is used correctly in most parts of the code, but in one place the check was against NULL instead.

Do I Need a Team of PhDs for This?

Formal verification is thought of as high-effort, expensive, and needing a large team of highly qualified experts. Our project shows that things are by far less bad than that, especially compared to other high-assurance methods. After the industry rule of thumb of $10k/LOC, Common Criteria EAL6 certification of seL4 would have cost about $87 million.

If we overestimate our effort with 30 person years and if we overestimate our fully loaded salary with $200k/year per person, we get $6 million spent. Even if you take into account that CC takes more than just providing evidence that the design and code works, our proof provides higher assurance than what EAL7 officially requires—EAL7 requires only formal design proofs, no formal implementation proofs. And it does that for an order of magnitude less money.

So, yes, it is still considerable effort, but it is at the same level required of normal, good quality design and implementation, not in the prohibitively expensive class anymore.

Do you need a team of PhDs for this? We didn't. Most of the verification engineers in the project did not have a PhD. Some, but not all, were PhD students. Many were never involved in theorem proving before. They were university graduates and they are certainly very smart people, but they learned machine-checked theorem proving on the job. This attests that modern theorem proving tools like Isabelle/HOL are mature enough to be used, even if they can undoubtedly still be improved. You will probably want at least one expert with previous experience on the team, and you will definitely want domain experts such as OS designers, but formal foundational verification on real code with about 10,000, maybe 50,000 LOC, for full functional correctness is definitely possible with today's technology and tools. It's fun, too. More people should get into it.

REFERENCES

[1] L4 microkernel: http://l4hq.org.

[2] The Isabelle theorem prover: http://isabelle.in.tum.de/, 2009.

[3] The L4.verified project: http://ertos.nicta.com.au/research/l4.verified/, 2009.

[4] R.J. Feiertag and P.G. Neumann, “The Foundations of a Provably Secure Operating System (PSOS),” in *AFIPS Conference Proceedings: 1979 National Computer Conference* (AFIPS Press, 1979), pp. 329–334.

[5] G. Heiser, “Your System Is Secure? Prove It!” USENIX *;login:*, vol. 32, no. 6, pp. 35–38, December 2007.

[6] G. Heiser, “Q: What is the difference between a microkernel?” http://www.ok-labs.com/blog/entry/singularity-vs-l4-microkernel-performance/, 2009.

[7] G.C. Hunt and J.R. Larus, “Singularity: Rethinking the Software Stack,” *ACM Operating System Review*, vol. 41, no. 2, 2007, pp. 37–49.

[8] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, “seL4: Formal Verification of an OS Kernel,” in *22nd ACM Symposium on Operating Systems Principles*, Big Sky, MT, USA, October 2009, pp. 207–220.

[9] T. Ormandy and J. Tinnes, “Linux Null Pointer Dereference due to Incorrect proto_ops Initializations,” 2009: http://archives.neohapsis.com/archives/fulldisclosure/2009-08/0174.html.

[10] G.J. Popek, M. Kampe, C.S. Kline, and E. Walton, “UCLA Data Secure Unix,” in *AFIPS Conference Proceedings: 1979 National Computer Conference* (AFIPS Press, 1979), pp. 355–364.

PERRY METZGER, WILLIAM ALLEN SIMPSON, AND PAUL VIXIE

improving TCP security with robust cookies

Perry E. Metzger is the managing partner of Metzger, Dowdeswell & Co. and is also pursuing a doctorate in computer science at the University of Pennsylvania. He desperately needs more sleep.

perry@piermont.com

William Allen Simpson is a very independent consultant, involved in design, implementation, and operation of Internet routing, network security, network protocols, wireless networking, game networking, real-time data collection and distribution, and many other projects for over 30 years.

William.Allen.Simpson@GMail.com

Paul Vixie took over BIND maintainance after Berkeley gave it up in 1989. He rewrote it (BIND8, 1995) and then hired other people to rewrite it again (BIND9, 2000). He has recently hired a new team to rewrite it yet again (BIND10, 201?). Between BIND releases, Paul founded the first neutral commercial Internet exchange (PAIX, 1998) and spent a small fortune fighting lawsuits by spammers (MAPS, 1998).

vixie@isc.org

Some men dream of fortunes, others dream of cookies.

—*fortune cookie*

THERE'S AN IMPENDING CRISIS, DRIVEN by the deployment of Domain Name System (DNS) Security (DNSSEC). DNS responses no longer fit in small UDP datagrams and are subsequently repeated in TCP sessions. We urgently need to robustly handle high rates of short-lived transactional TCP traffic and seamlessly segue to longer-lived sessions. TCP Cookie Transactions solve these problems and some denial-of-service attacks against TCP as well.

Current TCP implementations store enormous amounts of internal state for every connection. Heavily loaded servers can run out of memory and other resources simply by receiving too many connections (or bogus connection attempts) too quickly. TCP Cookie Transactions (TCPCT) deter spoofing of client connections and prevent server resource exhaustion by eliminating the need to maintain server state during establishment and after termination of connections. The TCPCT cookie exchange itself may optionally carry <SYN> data, limited in size to inhibit denial-of-service (DoS) attacks.

Motivation

Common DNSSEC-signed responses are as long as 1749 bytes. During key rollover, the response could be more than twice that size, much larger than the default UDP data size of 512 bytes.

Large DNS replies over UDP permit an attacker to amplify a denial-of-service attack. By spoofing a DNS request from a victim's IP address, an attacker can turn relatively short queries over a low bandwidth connection into a far more devastating amplification attack. That's potentially more potent than a generic attack, as operators cannot filter root server responses. Currently, only 2% of DNS root server queries over UDP are legitimate [33].

DNSSEC over UDP results in multiple IP fragments where the UDP headers and port numbers are only present in the first fragment. Badly implemented middleboxes [6]—such as stateless firewalls and network address translators (NAT) [28]—either drop all the fragments or pass the first and block the rest.

A horrific number of badly implemented middleboxes rewrite DNS over UDP messages according to local policy. These middleboxes assume that packets are not fragmented. Such middleboxes are likely to remain in place for many years.

UDP has no reliable signal that large datagrams won't work. Often the only symptom is a timeout, without any hint about which of the many possible problems occurred.

The burden is on DNS resolvers to try a protocol with less interference from middleboxes. Therefore, DNS resolvers repeat the same query over TCP.

Figure 1 shows that standard TCP creates server state immediately and retains it after the connection has closed during the TCP TIME-WAIT interval (usually 4 minutes).

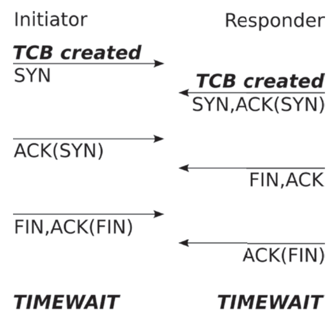


FIGURE 1: WHEN A ◄SYN► IS RECEIVED, CURRENT SERVER IMPLEMENTATIONS CREATE STATE (THE TCB) AND MAINTAIN THAT STATE UNTIL THE TIME-WAIT INTERVAL HAS EXPIRED (USUALLY 4 MINUTES).

Unfortunately, existing traffic patterns indicate that repeating most DNSSEC root UDP queries again over TCP would dramatically increase server load. After DNSSEC deployment in one major top-level domain, a 600% increase in TCP requests was reported [32] at the North American Network Operators Group (NANOG) meeting of June 2009.

To avoid overload, some operators are turning off the TCP port for DNS. That violates underlying DNS protocol expectations [2]. The inability to use TCP after missing or truncated UDP responses will prevent successful DNSSEC deployment.

TCP Cookie Transactions (TCPCT) permit TCP to be used in place of UDP for high-transaction-rate services without burdening servers. Operators can mitigate load by selective rejection of connection attempts without the Cookie option. Moreover, using the cryptologically secure robust cookie mechanism instead of UDP prevents the exploitation of amplification and fragmentation DoS attacks.

Robust Cookies

In 1994, Phil Karn described a mechanism to avoid accumulating server state during an initial protocol handshake. The client sends an opaque anti-clogging token (a "cookie"). The server responds to each communication attempt by issuing its own cookie that is dependent on the client cookie, and it retains no state about the attempt.

The client returns this pair of cookies to the server, demonstrating a complete communications path. If the client fails to reply, the server has no state to free.

Karn and Simpson set forth explicit design criteria:

> The computing resources themselves must also be protected against malicious attack or sabotage. . . . These attacks are mitigated through using time-variant cookies, and the elimination of receiver state during initial exchanges of the protocol. [15, pp. 2–3]
>
> It MUST NOT be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity. This implies that the issuing entity will use local secret information in the generation and subsequent verification of a cookie. [15, p. 12; 16, p. 19]
>
> The Responder secret value that affects its cookies MAY remain the same for many different Initiators. However, this secret SHOULD be changed periodically to limit the time for use of its cookies (typically each 60 seconds). [16, p. 20]

This use of the term *cookie* should not be confused with other uses of "cookie" or "magic cookie," such as by HTTP or X Window systems, and other security protocol attempts [27]. Each of these is missing one or more of the requirements: (1) eliminating responding server state; (2) using a local secret; (3) having a time limit.

Previous Papers and Proposals

Over the past 35 years, hundreds (perhaps thousands) of articles, papers, and reports have described network attacks using TCP: address and port spoofing, amplification, fragmentation, resource exhaustion, and others less commonly publicized [12]. Various incremental approaches have been proposed.

T/TCP [4] permits lightweight TCP transactions for applications that traditionally have used UDP. However, T/TCP has unacceptable security issues [13, 26].

By September 1996, the long anticipated DoS attacks in the form of TCP SYN floods were devastating popular (and unpopular) servers and sites. Phil Karn informally mentioned adapting anti-clogging cookies to TCP. Perry Metzger proposed adding Karn's cookies as part of a "TCP++" effort [22], and two years later as part of a "TCPng" discussion [23].

Daniel J. Bernstein implemented "SYN cookies," small cookies embedded in the TCP SYN initial sequence number. This technique was exceptionally clever, because it did not require cooperation of the remote party and could be deployed unilaterally. However, SYN cookies can only be used in emergencies; they are incompatible with most TCP options. As there is insufficient space in the sequence number, the cookie is not considered cryptologically secure. The SYN cookie mechanism remains inactive until the system is under attack, and thus is not well tested in operation. Because of these deficiencies, SYN cookies were not accepted for publication in the Internet Engineering Task Force (IETF) RFC series until recently [7].

In 1999, Faber, Touch, and Yue [9] proposed using an option to negotiate the party that would maintain TIME-WAIT state. This permits a server to entirely eliminate state after closing a connection.

In 2000, the Stream Control Transmission Protocol (SCTP) [29] was published with a mechanism partially based on Karn's ideas. There have been a number of barriers to deployment of SCTP [15].

In 2006, the Datagram Congestion Control Protocol (DCCP) [18] was published with a mechanism analogous to SYN cookies.

Medina, Allman, and Floyd [21] found that the vast majority of modern TCP implementations correctly handle unknown TCP options passing through middleboxes. A new TCP option sent in <SYN> and returned in <SYN,ACK(SYN)> will reliably indicate that both parties understand the extension. But it is still prudent to follow the [RFC 793] "general principle of robustness: be conservative in what you do, be liberal in what you accept from others."

Solving <SYN> Spoofing

The initial TCP <SYN> exchange is vulnerable to forged IP addresses, predictable ports, and discoverable sequence numbers [25]. A complete fix requires that IP sources be checked as they enter the provider network, ensuring that they match those assigned to the provider's customers. Unfortunately, this ingress-filtering best current practice [11] is not widely enforced, and source address forged attacks continue at growing rates.

TCP Cookie Transactions (TCPCT) bolster the defense against such attacks. A cookie option is exchanged as the connection is opened. These cookies are larger and more unpredictable than addresses, ports, sequence numbers, and timestamps. They validate the connection between two parties.

Figure 2 demonstrates the TCPCT cookie exchange.

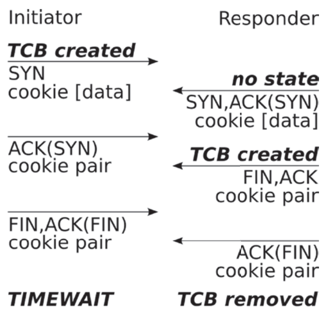


FIGURE 2: A NEW TCP OPTION CARRIES THE COOKIES, FOLLOWED BY OPTIONAL TRANSACTION DATA IN THE INITIAL EXCHANGE (◂SYN▸ AND ◂SYN,ACK▸).

AMPLIFICATION ATTACKS

TCP does not have the amplification problems of UDP [31]. A falsified source address on a <SYN> query results in a <SYN,ACK> response that is usually the same size as the query.

Unlike SCTP, TCPCT cookies are the same size in each direction, so the cookies themselves do not provide amplification.

Both T/TCP and a more recently proposed option [19] allow data to be carried on the <SYN,ACK> response, potentially allowing amplification. TCPCT enables sending this limited amount of data, as seen in Figure 2.

However, this optional feature is off by default, and is only enabled by an application on a per-port basis. Moreover, the feature may be temporarily disabled during periods of congestion and/or other resource limitations, transparently returning to default TCP behavior.

FRAGMENTATION FAILURES

Problems with IP fragmentation have long been well known [17]. For example, IP fragmentation doesn't work reliably and, more importantly, doesn't fail reliably. UDP has no segmentation and relies entirely on unreliable IP for fragmentation support.

TCPCT requires the TCP Timestamps Option [5], which in turn requires Path MTU Discovery [24] and that the Don't Fragment (DF) bit is always set in the IP header.

PORT PROBLEMS

Busy servers that deal with a large number of short transactions can experience port exhaustion.

For example, a Network Address Translator (NAT) maps routed hosts to its address, commonly implemented by assigning each connection to a different port [28]. When many hosts behind a NAT communicate with a common server, a port number must be assigned to each transaction. If too many transactions happen in rapid succession, the NAT will run out of port numbers.

DNS caching resolvers provide another example. When many hosts make queries through a caching resolver to a common server, a port number must be assigned to each transaction. If too many queries happen in rapid succession, the resolver will run out of port numbers. Repeated querying and aggressive retransmission [20] exacerbate these problems.

A closed TCP port must not be reused until a (TCP TIME-WAIT) timeout period has expired. If old port numbers are recycled too quickly, messages intended for the closed session cannot be distinguished from a newly opened session, appearing to be delayed duplicate transmissions.

TCPCT obviates antique duplicate transmissions by entirely eliminating server state after the <FIN> exchange. Only the client retains prior connection state for the required TCP TIME-WAIT period (see Figure 2).

TCPCT also handles reusing prior port numbers, by defining procedures that safely emulate persistent connections. Cookies and timestamps easily differentiate new sessions.

Most applications already follow the end-to-end principle and use the TCP close only as an optimization. Their data format provides all the necessary semantics for their needs.

TCPCT treats any closing <FIN> as advisory until it has been acknowledged by both parties. Like the <SYN>, each <FIN> is accompanied by the session cookies and timestamps. This inhibits a connection assassination attack with <FIN>.

RESOURCE RECYCLING

When a TCP <SYN> arrives with an unreachable source address, the target reserves transmission control block (TCB) resources and waits for a response to its <SYN,ACK>. These are called half-open connections. An attacker can repeatedly open connections with bogus source addresses, causing a target to retain state for each half-open connection until there are no resources for legitimate connections.

Moreover, busy servers that deal with a large number of short transactions can have legitimate problems with TCB exhaustion. If the number of different clients connecting to a server locks up too much server memory, then persistent connections will make the problem worse.

Using a different strategy, attackers need only open some long-running Initiators that do nothing or do things very slowly as a different form of DoS attack. TCPCT works with the TCP User Timeout Option [8] to limit accumulation of inactive connections.

TCPCT ameliorates TCB exhaustion by eliminating server state during the <SYN> exchange and again after the <FIN> exchange. Optional <SYN> data entirely eliminates TCB state for short transactions. After the connection has closed, state is retained only by the client for the required TCP TIME-WAIT period (see Figure 2, above).

TERMINATION TROUBLES

Perhaps the greatest security vulnerability of TCP itself is using an error indication (<RST>) to affect the operation of the protocol. This leads to TIME-WAIT assassination by antique duplicates [3], and connection assassination by third parties [10, 30].

TCPCT treats <RST> as advisory. Like the <SYN> and <FIN>, each <RST> is accompanied by the session cookies and timestamps. This inhibits a connection assassination attack with <RST>.

While cookies prevent most spoofed assassination attacks, the initial <SYN> exchange is particularly vulnerable. An attacker that can guess other fields could send a <RST> before the Responder <SYN,ACK> arrives with a proper cookie. The Initiator will not know about the attack.

Furthermore, cookies cannot defend against monkey-in-the-middle (MITM) attackers—where an attacker can record and/or reflect cookie, sequence, and timestamp values. Figure 3 demonstrates a standard TCP <SYN> assassination using <RST>.

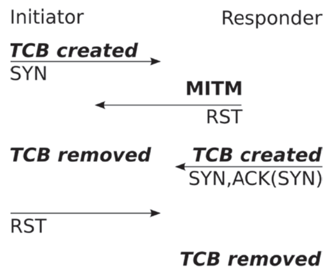


FIGURE 3: USING AN INJECTED <RST> PACKET TO ASSASSINATE A TCP CONNECTION

Therefore, receipt of <RST> has no effect on the operation of the protocol. All <RST> segments are merely counted [1, sec. 1.2.3]. Transmission will continue until a timeout expires [1, secs. 4.2.2.20(h), 4.2.3.5]. Arguably, this is the only substantial TCPCT change in TCP semantics.

Summary and Exhortation

TCP Cookie Transactions (TCPCT) provide a cryptologically secure mechanism to guard against simple flooding attacks sent with bogus IP sources or TCP ports.

TCPCT entirely eliminates server state during connection establishment and after termination, and it inhibits premature closing of connections. Also, implementations may optionally exchange limited amounts of transaction data during the initial cookie exchange, reducing round trips in short transactions.

Finally, implementations may optionally rapidly recycle prior connections. For otherwise stateless applications, this transparently facilitates persistent connections and pipelining of requests over each connection, reducing network latency and host task context switching.

We expect soon to have TCPCT implementations tested and deployed in some root and TLD servers. As caching resolvers and clients are updated, the load on servers should decrease.

Gentle reader, we hope that the numerous benefits of TCPCT will inspire you to request implementation and deployment on your favorite systems.

ACKNOWLEDGMENTS

Many thanks to Ted Faber, J. Bruce Fields, Fernando Gont, Craig Partridge, David P. Reed, Joe Touch, and other participants at the end2end-interest and namedroppers mailing lists for helpful comments on this topic.

REFERENCES

[1] R. Braden, ed., “Requirements for Internet Hosts—Communication Layers,” STD 3, RFC 1122, October 1989: http://tools.ietf.org/html/rfc1122.

[2] R. Braden, “Requirements for Internet Hosts—Application and Support,” STD 3, RFC 1123, October 1989. See section 6.1.3.2: http://tools.ietf.org/html/rfc1123.

[3] R. Braden, “TIME-WAIT Assassination Hazards in TCP,” RFC 1337, May 1992: http://tools.ietf.org/html/rfc1337.

[4] R. Braden, “T/TCP—TCP Extensions for Transactions—Functional Specification,” RFC 1644, July 1994: http://tools.ietf.org/html/rfc1644.

[5] V. Jacobson, R. Braden, and D. Borman, “TCP Extensions for High Performance,” RFC 1323, May 1992: http://tools.ietf.org/html/rfc1323. Updating: work in progress, March 4, 2009: http://tools.ietf.org/html/draft-ietf-tcpm-1323bis-01.

[6] B. Carpenter and S. Brim, “Middleboxes: Taxonomy and Issues,” RFC 3234, February 2002.

[7] W. Eddy, “TCP SYN Flooding Attacks and Common Mitigations,” RFC 4987, August 2007.

[8] L. Eggert and F. Gont, “TCP User Timeout Option,” RFC 5482, March 2009.

[9] T. Faber, J. Touch, and W. Yue, “The TIME-WAIT State in TCP and Its Effect on Busy Servers,” IEEE INFOCOM 99, pp. 1573–1584.

[10] S. Floyd, “Inappropriate TCP Resets Considered Harmful,” BCP 60, RFC 3360, August 2002.

[11] P. Ferguson and D. Senie, “Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing,” BCP 38, RFC 2827, May 2000.

[12] F. Gont, "Security Assessment of the Transmission Control Protocol (TCP)," February 2009: https://www.cpni.gov.uk/Docs/tn-03-09-security-assessment-TCP.pdf.

[13] C. Hannum, "Security Problems Associated with T/TCP," unpublished work in progress, September 1996: http://www.mid-way.org/doc/ttcp-sec.txt.

[14] Internet Engineering Task Force (IETF), "Intellectual Property Rights Disclosures": https://datatracker.ietf.org/ipr/.

[15] P. Karn and W. Simpson, "The Photuris Session Key Management Protocol," March 1995: draft-karn-photuris-01.txt.sp. Published as "Photuris: Design Criteria," in *Proceedings of Sixth Annual Workshop on Selected Areas in Cryptography*, LNCS 1758, (Springer-Verlag, August 1999).

[16] P. Karn and W. Simpson, "Photuris: Session-Key Management Protocol," RFC 2522, March 1999.

[17] C. Kent and J. Mogul, "Fragmentation Considered Harmful," 1987: http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-3.pdf.

[18] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, March 2006.

[19] A. Langley, "Faster Application Handshakes with SYN/ACK Payloads," work in progress, August 5, 2008: http://tools.ietf.org/html/draft-agl-tcpm-sadata-01.

[20] M. Larson and P. Barber, "Observed DNS Resolution Misbehavior," BCP 123, RFC 4697, October 2006.

[21] A. Medina, M. Allman, and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes," *Proceedings of the 4th ACM SIGCOMM/USENIX Conference on Internet Measurement*, October 2004: http://www
.icsi.berkeley.edu/pubs/networking/tbit-Aug2004.pdf.

[22] P. Metzger, "Re: SYN floods (was: does history repeat itself?)," September 9, 1996: http://www.merit.net/mail.archives/nanog/1996-09/msg00235.html.

[23] P. Metzger, "Re: what a new TCP header might look like," May 12, 1998: ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail.

[24] J. Mogul, and S. Deering, "Path MTU Discovery," RFC 1191, November 1990.

[25] R. Morris, "A Weakness in the 4.2BSD Unix TCP/IP Software," Technical Report CSTR-117, AT&T Bell Laboratories, February 1985: http://pdos.csail.mit.edu/~rtm/papers/117.pdf.

[26] route [at] infonexus [dot] com, "T/TCP vulnerabilities," *Phrack Magazine*, vol. 8, no. 53, July 8, 1998: http://www.phrack.org/issues.html?issue=53&id=6.

[27] W. Simpson, "IKE/ISAKMP Considered Harmful," USENIX *;login:*, December 1999: http://www.usenix.org/publications/login/1999-12/features/harmful.html.

[28] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, January 2001.

[29] R. Stewart, ed., "Stream Control Transmission Protocol," RFC 4960, September 2007.

[30] J. Touch, "Defending TCP against Spoofing Attacks," RFC 4953, July 2007.

[31] R. Vaughn and G. Evron, “DNS Amplification Attacks,” March 17, 2006: http://www.isotf.org/news/DNS-Amplification-Attacks.pdf.

[32] D. Wessels, “DNSSEC, EDNS, and TCP,” June 2009: http://www.nanog.org/meetings/nanog46/presentations/Wednesday/wessels_light_N46.pdf.

[33] D. Wessels and M. Fomenkov, “Wow, That’s a Lot of Packets,” *Proceedings of the Passive and Active Measurement Workshop (PAM),* April 2003: http://www.caida.org/publications/papers/2003/dnspackets/wessels-pam2003.pdf.

DAVID N. BLANK-EDELMAN

practical Perl tools: essential techniques

David N. Blank-Edelman is the director of technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs.

dnb@ccs.neu.edu

EVERY ONCE IN A WHILE IN THIS COLUMN I like to get meta. In the past we have talked about ways to become better programmers. We've looked at tools and methodologies like test-first programming which force you into a working style that produces better programs. For this column, I'd like to share three tools that can help you become a better, or at least a more efficient, Perl programmer in particular. So, perhaps this month, we'll go half-meta.

Being Strict

I know that some percentage of my readership is going to roll their eyes with such vigor that you can hear the noise they make in their sockets, but I must start with this tip. So go ahead, get it out of your system now because I'm going to say it:

```
use strict;
```

The eye rolling comes because the Perl community has been chanting "use strict;! use strict;! use strict;!" to itself like some scene from *Eyes Wide Shut* for many, many years now. When you turn strict on for a program, the Perl interpreter will complain about a whole host of potential issues with your program that go a bit beyond syntax errors. The complaints range from simple things such as

```
Name $blah used only once: possible typo
```

if a variable only appears once in a program (for example, it is set, but never read from—that's often a sign that there's a typo in the name) to more sophisticated warnings such as

```
Global symbol $blah requires explicit package name
```

which are trying to strong-arm you into using local variable scopes (since larger programs that use all global variables are fragile and easily broken).

The reason I mention this tip at all, given how pervasive it is in the community, is that I know it took me a while to get "use strict;" religion. I suspect there are others who have lapsed in the same manner. I think there are two main reasons why people get turned off early in their programming career by this pragma and never really come back to using it as a matter of course (i.e., circumstances don't demand otherwise):

1. It yammers so. Sometimes people new to the language get overwhelmed by the quantity of error messages, especially the more cryptic ones.

This turns them off early in their Perl programming learning curve, and they never really gain a desire to be yelled at by the interpreter ("Thank you, Sir, may I have another error message? Thank you, Sir. May I have another?"). The good news is that Perl developers have worked diligently over the years to make the production of the error messages smarter and the messages themselves more comprehensible. There is also a perldiag documentation section (perldoc perldiag) that provides at least a smidgen more information for every single error message the core Perl interpreter might emit, thus making them more helpful. If you shied away from strict mode before for this reason, I'd encourage you to try it again and see if it works better for you.

2. Some of the error messages that `strict` mode emits require the spankin' new programmer to understand some programming concepts that may initially be beyond their comprehension. I'm thinking specifically of the scoping-related error message I mentioned before of Global symbol $blah requires explicit package name, which comes up a great deal in first-effort programs. The perldiag reference page I mentioned before says this about it:

   Global symbol "%s" requires explicit package name
   (F) You've said "use strict" or "use strict vars", which indicates that all variables must either be lexically scoped (using "my"), declared beforehand using "our", or explicitly qualified to say which package the global variable is in (using "::").

It is a very direct and pointed explanation with a teaser about how you might fix the problem, but it only describes one or two trees of the forest the programmer is likely to be lost in at that point. If you aren't familiar with lexical and global scoping in programs or are just not clear on Perl's particular way of manifesting these concepts, getting a bunch of these error messages is not going to help much even with this explanation. There's not a lot I can suggest for this case except that the programmer find a text that explains "my", "local", and "our" in a way that makes sense to them before starting to use `strict` mode.

Before we move on I just want to mention a couple of ways that the "use strict;" idea has been extended:

3. The module Acme::use::strict::with::pride describes itself as performing this service: "enforce bondage and discipline on very naughty modules" and says:

   using Acme::use::strict::with::pride causes all modules to run with use strict; and use warnings;

   Whether they like it or not :-)

In general I don't advocate forcing your choices about how strict a programmer should be on others, but perhaps you have a reason to make sure all of the code you are running passes a "use strict;" test.

4. There are modules like Tie::StrictHash which allow you to subvert the usual auto-vivification nature of hashes (i.e., if you reference a hash key that didn't exist before, perhaps because of a typo, it comes into being whether you wanted that to happen or not). As the docs say:

   Tie::StrictHash is a module for implementing some of the same semantics for hash members that use strict gives to variables. The following constraints are applied to a strict hash:

   - No new keys may be added to the hash except through the add method of the hash control object.

- No keys may be deleted except through the delete method of the hash control object.
- The hash cannot be re-initialized (cleared) except through the clear method of the hash control object.
- Attempting to retrieve the value for a key that doesn't exist is a fatal error.
- Attempting to store a value for a key that doesn't exist is a fatal error.

This sort of discipline can be helpful in all sorts of situations.

Being Tidy

Let's leave all of that kink-themed programming discussion behind for the moment and move on to the question of why your parents were always after you to clean your room. You may have ignored the tool we're going to talk about in this section because it seemed like an aesthetic nicety, but I hope to convince you otherwise. There's a lovely module called Perl::Tidy that comes with a command-line tool called "perltidy." perltidy takes in your code and reformats it to match a set of predefined (by you) stylistic conventions. It's similar to the C program source formatting called "indent" but custom honed for Perl source code. Perl code that has been run through perltidy looks neater and (depending on your stylistic preferences) more readable. Given that Perl is a bit of a punctuation parking lot with a not quite deserved reputation (from those who have never seen APL) of looking like line noise, this can be a considerable improvement.

Let's look at perltidy in action so I can explain why you should be running all of your code through it even during the process of writing it. Here's an example from the perltidy home page (http://perltidy.sourceforge.net/) that shows the most dramatic sort of improvement:

```
%TV=(flintstones=>{series=>"flintstones",nights=>[qw(monday thursday
friday)],
members=>[{name=>"fred",role=>"lead",age=>36,},{name=>"wilma",role=
>"wife",
age=>31,},{name=>"pebbles",role=>"kid",age=>4,},],},jetsons=>{series=>"j
etsons",
nights=>[qw(wednesday saturday)],members=>[{name=>"george",role=>"le
ad",age=>41,
},{name=>"jane",role=>"wife",age=>39,},{name=>"elroy",role=>"kid",ag
e=>9,},],},
simpsons=>{series=>"simpsons",nights=>[qw(monday)],members=>[{name
=>"homer",
role=>"lead",age=>34,},{name=>"marge",role=>"wife",age=>37,},{name=>
"bart",
role=>"kid",age=>11,},],},);
```

run through perltidy becomes this:

```
%TV = (
  flintstones => {
      series  => "flintstones",
      nights  => [qw(monday thursday friday)],
      members => [
          {
              name => "fred",
              role => "lead",
              age  => 36,
            },
            {
                name => "wilma",
                role => "wife",
                age  => 31,
            },
            {
                name => "pebbles",
                role => "kid",
                age  => 4,
            },
        ],
    },
    jetsons => {
        series  => "jetsons",
        nights  => [qw(wednesday saturday)],
        members => [
            {
                name => "george",
                role => "lead",
                age  => 41,
            },
            {
                name => "jane",
                role => "wife",
                age  => 39,
            },
            {
                name => "elroy",
                role => "kid",
                age  => 9,
            },
        ],
    },
    ...
);
```

Hopefully you don't have to look at code (from other people, right?) that looks like the "before" in this example. But if you do get code from a colleague that isn't that easy to read, perltidy can help.

But this isn't the kind of result that makes perltidy essential. Here is a more interesting example:

```
sub hooberbloober {

   if (test_something()){
       if ($fred == 3){
          check_with_Shiva();
          # ... lots of code
   }}
   # ... lots of code
   do_the_dance_of destruction();
   # ... lots of code
   spin_the_wheel();
}
```

Why is this interesting? It helps demonstrate two reasons why you should hook perltidy into your editor (all of the major ones can do it) so you can run perltidy over code as you write it.

First, there's a class of errors that we have all run into at one time or another having to do with improperly placed closing brackets. It is especially easy to do in cases where the chunks of your code spans multiple screens. We have all had to debug code whose program flow didn't quite work as we anticipated because a section of code was put in or left out of a conditional block by mistake. In that last example, maybe we only wanted to do_the_dance_of_destruction() based on one of the conditional tests. If there was lots more ancillary code in our example, it might not be easy to see that we've closed an if() block prematurely. But if we run it through perltidy, the error jumps right out thanks to the reformatted indentation:

```
sub hooberbloober {

    if ( test_something() ) {
        if ( $fred == 3 ) {
            check_with_Shiva();
            # ... lots of code
        }
    }
    do_the_dance_of destruction();

    # ... lots of code
    spin_the_wheel();
}
```

Second, there's considerable value to always looking at and working with clean-looking code. It has a subtle but powerful effect on how you work. Here's a quote from an invited talk I gave at LISA '07 on what sysadmins could learn from professional cooks and others in the cooking world:

> I worked with a chef who used to step behind the line to a dirty cook's station in the middle of the rush to explain why the offending cook was falling behind. He'd press his palm down on the cutting board, which was littered with peppercorns, spattered sauce, bits of parsley, bread crumbs and the usual flotsam and jetsam that accumulates quickly on a station if not constantly wiped away with a moist side towel. "You see this?" he'd inquire, raising his palm so that the cook could see the bits of dirt and scraps sticking to the chef's palm, "That's what the inside of your head looks like now. Work clean!"
>
> —Anthony Bourdain in *Kitchen Confidential*

perltidy does an excellent job of helping you find small errors not caught by the interpreter's syntax checks and work clean.

Being Critical

OK, last tool. If you liked how "use strict;" provided feedback about problems with your code, then you are going to love this. Perl::Critic, and its accompanying command-line program perlcritic, goes even further in this direction. The documentation describes it as:

> an extensible framework for creating and applying coding standards to Perl source code. Essentially, it is a static source code analysis engine. Perl::Critic is distributed with a number of Perl::Critic::Policy modules that attempt to enforce various coding guidelines. Most Policy modules are based on Damian Conway's book Perl Best Practices. However, Perl::Critic is not limited to PBP

and will even support Policies that contradict Conway. You can enable, disable, and customize those Polices through the Perl::Critic interface. You can also create new Policy modules that suit your own tastes.

While I wouldn't necessarily run perlcritic over my code as often as I would perltidy, it is definitely helpful to periodically feed your code to perlcritic as you go along. To give you an idea of how it works, here's some output when run over some sample code found in this very column from 2006:

```
$ perlcritic geocode.pl:
Code before strictures are enabled at line 5, column 5.  See page 429 of PBP.
(Severity: 5)
```

The error here is I've not included (for space reasons) "use strict;" in my code. If I pick some other code I wrote back in 2005, it tells me about more interesting errors:

```
   chart.pl: Bareword file handle opened at line 20, column 1.  See pages
202,204 of PBP.  (Severity: 5)
   chart.pl: Two-argument "open" used at line 20, column 1.  See page 207 of
PBP.  (Severity: 5)
```

It's complaining about this line in the code:

```
open (T,">/tmp/t.png") or die "Can't open t.png:$!\n";
```

which is using conventions that have since fallen out of favor. A better way to write that would be:

```
open my $T, '>', '/tmp/t.png' or die "Can't open t.png:$!\n";
```

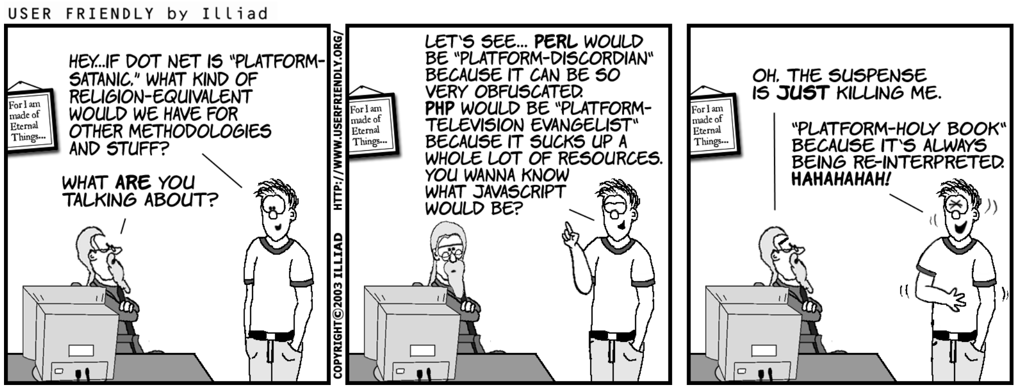
which passes perlcritic (with the default rules) with flying colors.

But the default settings for perlcritic only show the most flagrant violations. If I crank that up to 11 (or, rather, to a severity level of "brutal"), I get these errors from that one line:

```
Code is not tidy at line 1, column 1.  See page 33 of PBP.  (Severity: 1)
RCS keywords $Id$ not found at line 1, column 1.  See page 441 of PBP.
(Severity: 2)
RCS keywords $Revision$, $HeadURL$, $Date$ not found at line 1, column 1.
See page 441 of PBP.  (Severity: 2)
RCS keywords $Revision$, $Source$, $Date$ not found at line 1, column 1.
See page 441 of PBP.  (Severity: 2)
No "$VERSION" variable found at line 1, column 1.  See page 404 of PBP.
(Severity: 2)
Close filehandles as soon as possible after opening them at line 1, column 4.
See page 209 of PBP.  (Severity: 4)
Module does not end with "1;" at line 1, column 4.  Must end with a recogniz-
able true value.  (Severity: 4)
Code not contained in explicit package at line 1, column 4.  Violates encapsula-
tion.  (Severity: 4)
Code before strictures are enabled at line 1, column 4.  See page 429 of PBP.
(Severity: 5)
Code before warnings are enabled at line 1, column 4.  See page 431 of PBP.
(Severity: 4)
Magic punctuation variable used in interpolated string at line 1, column 41.  See
page 79 of PBP.  (Severity: 2)
Found "\N{SPACE}" at the end of the line at line 1, column 65.  Don't use
whitespace at the end of lines.  (Severity: 1)
```

and that's just with the default module rules. There are many other Perl::Critic::* modules on CPAN that can add even more or different fussiness. Clearly, much of what it is complaining about can be ignored (since I was only testing a single line), but in real life cases perlcritic often offers really helpful criticism. If you want to play with Perl::Critic without installing the module, some people in the Perl community have been kind enough to set up a Web site (http://perlcritic.com) that will audit your code for you remotely.

All three of the tools we've looked at in this column can, in the right measure, really help improve your Perl programming. Enjoy, and I'll see you next time.


USER FRIENDLY by Illiad
For I am made of Eternal Things...
HEY...IF DOT NET IS "PLATFORM-SATANIC," WHAT KIND OF RELIGION-EQUIVALENT WOULD WE HAVE FOR OTHER METHODOLOGIES AND STUFF?
WHAT ARE YOU TALKING ABOUT?
HTTP://WWW.USERFRIENDLY.ORG/
COPYRIGHT©2003 ILLIAD
LET'S SEE... PERL WOULD BE "PLATFORM-DISCORDIAN" BECAUSE IT CAN BE SO VERY OBFUSCATED. PHP WOULD BE "PLATFORM-TELEVISION EVANGELIST" BECAUSE IT SUCKS UP A WHOLE LOT OF RESOURCES. YOU WANNA KNOW WHAT JAVASCRIPT WOULD BE?
OH. THE SUSPENSE IS JUST KILLING ME.
"PLATFORM-HOLY BOOK" BECAUSE IT'S ALWAYS BEING RE-INTERPRETED. HAHAHAHAH!

PETER BAER GALVIN

Pete's all things Sun: swaddling applications in a security blanket

Peter Baer Galvin is the CTO of IT Architecture for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is co-author of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter blogs at http://www.galvin.info and twitters as "PeterGalvin."

pbg@cptech.com

THE IMMUTABLE SERVICE CONTAINERS (ISC) project seeks to increase systemic security within Solaris. An ISC is a potentially perfect locale in which to run applications where increased security is desired. Within the ISC ecosystem is the ability to clone ISCs and reset them to a known good state, and the potential for automatic actions in case of a security incident. ISC is not currently an integrated or support project, but it is an important security step for Solaris and therefore worth discussing even at this early stage.

Immutable Service Containers

ISC currently consists of a plan and documents, as well as the "OpenSolaris Immutable Service Container construction kit" [1], a set of tools for building ISCs within OpenSolaris. The project's goal is to go beyond OpenSolaris, creating, for example, an ISC from a VirtualBox virtual machine. Fundamentally, ISCs are a set of tools, steps, and techniques that can be used to more securely run applications in highly managed environments and can be set up and used wherever the operating system or virtualization tools provide the features required. According to their definition, "ISCs provide a security-reinforced container into which a service or set of services is deployed."

The design goals for ISCs include limiting exposure by reducing services and using resource controls to run those that are required, limiting change by making service and critical operating environment configuration read-only, limiting rights based on the least privilege model, and increasing integrity by isolating the service for monitoring and enforcement [2]. There are several benefits to deploying infrastructure based on ISCs, including a more secure starting point for deployment and management, automation of application deployment, built-in best practice security aspects, decreased chance of break-ins, decreased chance of damage from a break-in, and more likely standardization of security within the infrastructure.

The reasoning behind the ISC project is that, even when the "right" steps to increase security are known, they are infrequently followed and even less frequently checked and updated. An ISC has all of those right steps already integrated, easing the effort needed to secure an application's environment. Certainly the world would be a better place if all applications were run in ISCs, but the first steps are designing, testing, and documenting them. My hope for this column is that it brings attention to ISCs and helps to encourage their propagation.

ISC is a core building block of some security initiatives at Sun and security trends in general. It can be part of an adaptive security architecture [3] which responds to threats quickly while minimizing potential damage. Further, ISCs can be part of an autonomic security layer that can be self-cleansing, self-updating, and can automatically roll back and quarantine its components, even performing a self-assessment or self-destruction if needed.

The ISC Architecture consists of an ISC "dock" and the ISC itself. The dock provides security enforcement, monitoring, resource controls, and other management functions. It communicates via SSH to one or more ISCs. The ISC consists of the security-hardened container (be it a Solaris container, a virtual machine, or other similar structure), plus security functions.

Hands-on

Even though the ISC project states that it is more of a proof-of-concept than a production-ready service, I thought it would be interesting to take the current implementation for a test drive. Currently, ISCs can be implemented on OpenSolaris via Solaris Zones (a.k.a. "containers"). Other options will be possible in the future (as discussed in the next section). Implementation and configuration of ISCs currently take a few steps, from downloading and running the scripts through editing firewall configuration files. The steps are outlined in the project wiki [4], but I include them here and add some explanation as well as some testing results. If you want to save some steps, you can download a virtual machine image of OpenSolaris already configured with an ISC.

The ISC description so far is certainly high on promise. Unless components or implementations are released officially and supported by Sun, it will be difficult to judge how effectively ISC meets its goals. The current pre-release should give a good indication of how close ISCs are to production usability, how easy they will be to operate, and how close they are to delivering on their promise.

For the purposes of this column I tested the OpenSolaris V 1.0 preview of ISC. At this point ISC is not even an OpenSolaris package. Rather, it's available as a Mercurial (source code management) repository. I started from a fresh copy of OpenSolaris 2009.06. For a shortcut, a prebuilt OpenSolaris containing ISC and an ISC container can be downloaded in OVF format and run as a VM guest inside of an OVF-format supporting virtual machine manager (such as VirtualBox).

First, Mercurial must be installed, and the Mercurial repository containing ISC downloaded:

```
opensolaris$ pfexec pkg install SUNWmercurial
. . .
opensolaris$ hg clone https://kenai.com/hg/isc-source isc
. . .
```

Next, the configuration script is run to modify the system and create an ISC:

```
opensolaris$ pfexec isc/bin/iscadm.ksh
Setting netmask of isc0 to 255.255.255.0
Installing SMF method: /lib/svc/method/svc-isc-enc-swap
Installing SMF manifest: /var/svc/manifest/site/isc-enc-swap.xml
Installing SMF method: /lib/svc/method/svc-isc-enc-scratch
Installing SMF manifest: /var/svc/manifest/site/isc-enc-scratch.xml
isc1: No such zone configured
Use 'create' to begin configuring a new zone.
A ZFS file system has been created for this zone.
   Publisher: Using opensolaris.org (http://pkg.opensolaris.org/release/).
       Image: Preparing at /export/isc/isc1/zone/root.
       Cache: Using /var/pkg/download.
Sanity Check: Looking for 'entire' incorporation.
  Installing: Core System (output follows)
DOWNLOAD                    PKGS          FILES       XFER (MB)
Completed                  20/20      3021/3021     42.55/42.55

PHASE                    ACTIONS
Install Phase          5747/5747
    Installing: Additional Packages (output follows)
DOWNLOAD                    PKGS          FILES       XFER (MB)
Completed                  37/37      5598/5598     32.52/32.52

PHASE                    ACTIONS
Install Phase          7329/7329

                 Note:  Man pages can be obtained by installing SUNWman
          Postinstall:  Copying SMF seed repository ... done.
          Postinstall:  Applying workarounds.
                 Done:  Installation completed in 373.243 seconds.

           Next Steps:  Boot the zone, then log into the zone console
                        (zlogin -C) to complete the configuration process
          Global zone version:  entire@0.5.11,5.11-0.111:20090514T145840Z
      Non-Global zone version:  entire@0.5.11,5.11-0.111:20090514T145840Z
                   Evaluation:  Packages in isc1 are in sync with global zone.
Attach complete.
          Global zone version:  entire@0.5.11,5.11-0.111:20090514T145840Z
      Non-Global zone version:  entire@0.5.11,5.11-0.111:20090514T145840Z
                   Evaluation:  Packages in isc1 are in sync with global zone.
Attach complete.
```

Installation transforms the OpenSolaris deployment from a general-use system to a much more secured environment. Even the boot name changes. The /etc/motd is changed to display a message about unauthorized use. And the GUI login is disabled in favor of command-line interactions. Clearly this should not be done on a desktop deployment of OpenSolaris—it's all about creating secure server environments in which to run services. A Solaris container called "isc1" is preinstalled, with a default password of "iscroot" that needs to be changed. Note that this is not a security hole, because there is no way to connect to the container from outside the system until services are enabled and the global zone is configured to allow communication to the ISC.

A boot environment cache update and reboot gets the system ready for use:

```
opensolaris$ pfexec bootadm update-archive
opensolaris$ pfexec shutdown -g 0 -i 0 -y
```

Once an ISC container is built, it has many interesting aspects. For example, the configuration is hardened, auditing enabled, and the stack set to non-executable. Also, as well as the usual default container file systems, there is a new /scratch one provided. This is a non-persistent encrypted file system that applications can use to securely store log files, temporary files, and other contents. Because ZFS does not yet implement encryption, there is some indirection involved in the implementation of the encrypted scratch space. Essentially, a ZFS zvol (volume) is the core, and then a LOFI (loopback file system) is created with encryption enabled (using an ephemeral key that will be lost when the system is shut down) and a zpool on top of that, with the end result of exporting a file system that is encrypted:

```
root@isc1:~# df -kh
Filesystem          size        used       avail  capacity       Mounted on
. . .
/scratch           63M         19K       63M      1%              /scratch
. . .
pbg@opensolaris:~$ zfs list
NAME                             USED     AVAIL     REFER    MOUNTPOINT
. . .
rpool/export/scratch             300M     3.31G     19K      /export/scratch
rpool/export/scratch/global      100M     3.31G     19K      /export/scratch/global
rpool/export/scratch/global/     100M     3.41G     1.19M    -
    scratch_file
rpool/export/scratch/isc1        100M     3.31G     19K      /export/scratch/isc1
rpool/export/scratch/isc1/       100M     3.41G     1.19M    -
     scratch_file
scratch-global                   71.5K    62.9M     19K      /scratch-global
scratch-isc1                     71.5K    62.9M     19K      scratch-isc1
. . .
pbg@opensolaris:~$ zpool status -v scratch-isc1
      pool:  scratch-isc1
     state:  ONLINE
     scrub:  none requested
    config:

              NAME            STATE       READ       WRITE      CKSUM
              scratch-isc1    ONLINE         0           0          0
                /dev/lofi/3   ONLINE         0           0          0
pbg@opensolaris:~$ lofiadm
Block Device       File                                Options
/dev/lofi/1        /devices/pseudo/zfs@0:1c            Encrypted
/dev/lofi/2        /devices/pseudo/zfs@0:2c,raw        Encrypted
/dev/lofi/3        /devices/pseudo/zfs@0:3c,raw        Encrypted
/dev/lofi/4        /devices/pseudo/zfs@0:5c,raw        Encrypted
```

Once an ISC is created, applications can be installed and enabled within it. From the ISC wiki comes the example of installing and enabling Apache:

```
opensolaris$ pfexec zlogin isc1 pkg install SUNWapch22
opensolaris$ pfexec zlogin isc1 svcadm enable apache22
```

Because the default is for no communication to be allowed to the ISC, the firewall rules much be changed to allow communication. This is done by editing /etc/ipf/ipf.conf and, if the IP address of the ISC guest is 192.168.0.1, adding a line such as:

```
pass in quick on e1000g0 proto tcp from any to 192.168.0.1 port = 80 keep state
```

Because by default the ISC's network is not accessible from outside the system, a new NAT rule has to be put in place to route traffic that reaches the system on port 80 into the ISC housing the Web server. Edit /etc/ipf/ipnat.conf and add a line such as:

```
    rdr e1000g0 0.0.0.0/0 port 80 -> 192.168.0.1 port 80
```

For those commands to take effect, the firewall must be told to reload its configuration files via:

```
opensolaris$ pfexec ipf -Fa -f /etc/ipf/ipf.conf
opensolaris$ pfexec ipnat -FC -f /etc/ipf/ipnat.conf
```

From a separate system, pointing a Web browser to the IP address of the host containing the ISC should allow connection to the secure Web server within the container.

The ISC infrastructure includes a new command-line script to manage and modify ISCs. For example, to create a new ISC, say ISC 2, the command line would be:

```
opensolaris$ pfexec isc/bin/iscadm.ksh -c -i -n 2
```

Currently, the iscadm script performs no other major actions. The plan is for it to control the creation of snapshots, deletion of ISC environments, verification that an ISC environment has not been modified, and so on. Such changes would be a welcome addition to the ISC functionality.

The Future

ISC is an active project with several steps likely in the future. Glenn Brunette, a Sun Distinguished Engineer, is leading the charge on this project and actively working on designing, implementing, and automating the creation of ISCs. Next steps potentially include the following areas:

- Updates to take advantage of new OpenSolaris functionality as it is integrated (such as ZFS encryption, Validated Execution, Always-On Auditing, and other projects).
- New reference configurations that can utilize VirtualBox as the containment model in place of OpenSolaris zones, allowing for the use of alternative guest operating systems beyond OpenSolaris. (For now, the project will likely continue with OpenSolaris as the host OS, due to the security feature set it provides.)
- New operational configurations that implement the autonomic security use cases [6].

The project is also giving consideration to more advanced configuration tools that allow a user to create virtual ISC networks (using Crossbow) [7]. Another area of interest is migration and validation tools to help people move their applications into ISCs and ensure that the security configuration is implemented properly. And on the practical front, there are plans to have ISCs available at some point on Amazon EC2 as an extension of Sun's security-enhanced OpenSolaris AMI efforts.

The project is also actively seeking input from Sun customers, which in turn creates RFEs (requests for enhancement) that get put into the development queue. If security is of interest at your site, downloading and using the current toolset and giving feedback as to what works, what doesn't, and what features you would like to see should be on your to-do list.

For more information on the ISC project, including discussion forums, publications, podcasts, and presentations, visit the project's home at http://kenai.com/projects/isc/pages/Home.

Conclusion

ISC has lofty goals, and the current non-production implementation meets quite a few of them. Assuming the project does move into production with a complete feature set, ISCs will be a huge leap forward for cloud computing and datacenter application deployment. Easy to use, high-potency security is the nirvana data managers seek but frequently don't find. ISCs could be one of those rare exceptions.

REFERENCES

[1] http://kenai.com/projects/isc/pages/OpenSolaris.

[2] http://kenai.com/projects/isc/pages/Architecture.

[3] Sun Adaptive Security Architecture Blueprint 820-6825.

[4] http://kenai.com/projects/isc/pages/OpenSolaris#Service_Installation_and_Confi.

[5] http://kenai.com/projects/isc/downloads.

[6] http://kenai.com/projects/isc/pages/Autonomic.

[7] http://kenai.com/projects/isc/pages/Networking.

DAVE JOSEPHSEN

iVoyeur: 7 habits of highly effective Nagios implementations

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

ONCE, SOME YEARS AGO, WHEN WE were all younger, I had something of a desire for knowledge (I'm guessing you can probably relate). Don't get me wrong: today I spend a great deal of my time learning, more time even than when I felt that urgent want of it, but that's not how I'd describe it anymore. Now it's more of a habit, a taste for learning that is tempered by the things I wish I hadn't learned, tempered that is, by the knowledge that haunts me. I'll give you an example.

I don't remember who, and I don't remember where or when, but somewhere, somewhen, someone taught me the novel writer's "first line" rule. That the single most important part of a novel was its first line. That it was obvious from line one whether a piece of fiction was in fact a piece of crap. Now, I don't write fiction (or haven't yet anyway), and really I don't even consider myself a "writer," but this "first line" business, well, it haunts me. It's something I simultaneously wish I hadn't learned and can't let myself ignore. If you weren't aware of it, I'm sorry to have passed it on to you. Even when writing 2000-word articles for a tech journal, I invariably obsess over that confounded first line, revising this word or that, deleting entirely and starting over from scratch until my opening paragraphs have nothing whatsoever to do with my eventual subject matter. I'll give you an example.

The opening line the computer geek part of me wanted to begin this article with was, "There is a fine line between monitoring systems that are effective and those that are annoying and useless." "A fine line!?" the lit-geek in me exclaims. "Why don't you just define a CLICHÉ variable and put a 'while(1)' around it? Or better yet, send two sleeping pills to the entire subscribed readership of *;login:* since that's the effect you're obviously going for?!" (My inner lit-geek is kind of an elitist jerk.) At this point in my process, I usually delete the line in favor of something absurd and unrelated, but in this case I couldn't bring myself to delete it.

A fine line is in fact what it is; so fine that any one of the seven tips I'm about to share can completely change the effectiveness of an otherwise maligned Nagios implementation. Further, all of them are common pitfalls that I myself have fallen victim to at one point or another, so they are things that are likely to help other sysadmins. First line be damned: this time I will have the courage to use a cliché if it is apt. This time I will NOT give myself over to the right half of my brain. This time, I REFUSE to write an unrelated yet entertaining introduction that requires a complex and clever segue into otherwise dry subject matter.

Oops.

1. Eliminate DNS Dependencies

The first tip I'd like to share has to do with name resolution. You can specify hosts in Nagios by IP or DNS name. It's probably a toss-up which of these is more reliable. Using names makes Nagios dependent on an operational DNS infrastructure. Using IPs eliminates the dependency but is a management nightmare; IPs will change, and Nagios won't be updated. In practical terms, using names gives you rarely occurring, large outages, while using IPs gives you more commonly occurring, individual outages.

You might think that Nagios being functional during a DNS outage would be the least of your worries, but you'd be wrong, in my humble opinion. If Nagios can remain functional during a DNS outage, it can provide good data on what boxes in the infrastructure actually ceased to function properly during the outage. There's a huge difference between a monitoring system that can provide good data during and especially after a cascading DNS failure and one that cannot. That's the kind of data that really lends credibility to the system, and much of the difference between effective implementations and maligned ones can be measured in credibility.

I recommend specifying names in the configuration files and then implementing a DNS cache and name resolver on the Nagios box itself. Set up zone transfers or otherwise automate the replication of DNS information from your real nameservers to the Nagios box. This solves all of our name-resolution woes; Nagios will point at itself for name resolution and won't rely on external DNS, while at the same time, there is no management overhead on keeping IPs up to date beyond the initial configuration.

We use djbdns [1] for this purpose, which I like very much. It's a small, lightweight, secure system that is easily implemented and updated.

2. Minimize Notifications

There are a couple of very large problems with monitoring systems that send too many notifications. The first is that the credibility of the monitoring system suffers when folks get notified about things they don't care about. It makes the system "seem" stupid, and that perception is going to make it difficult for you to get resources to improve the system.

The second, closely related problem is that people will begin to ignore the notifications. When actually important notifications are sent, they'll be ignored, and when/if management follows up, the monitoring system will be blamed for not sending notifications at all. The lack of credibility will make it easier for accusations like this to stick (despite evidence to the contrary). Further, if people don't trust the system, they'll be more likely to roll their own monitoring tools and less likely to ask you for help. This in turn will tend to magnify and compound the original perception until vendors are brought in and something truly stupid is implemented.

I'm not saying that you shouldn't *monitor* lots of services. I'm only saying that you should refrain from notifying anyone other than yourself about anything unless:

- they've specifically asked you to do it
- there's an SLA around it
- there's a policy requiring it

Even if one or more of these requirements has been satisfied, I'd offer them some alternatives instead (wouldn't a daily/hourly report of boxes with high CPU utilization be better?). Further, all notifications should be based on thresholds that you've performed some analysis to obtain. Holt-Winters forecasting [2] is superb for this sort of thing, but anything is better than nothing. The worst thing you can do is make up some arbitrary thresholds (or, worse, take theirs), create a notification group that includes everybody, and turn on Nagios (even if they ask you to).

There's a creeping entropy about this problem that makes it seem more innocuous than it is; notifications won't be ignored on the first or second day. Things will just slowly get progressively worse degree by tiny degree until everyone's pager is full of meaningless crap, no one notices real outages occurring, and the vendors arrive. You really need to stay on top of useless notifications. Hunt them down and eliminate them on a regular basis, ask people if they care about the notifications they're getting, see if you can get a policy setup that requires problem acknowledgments for every notification, etc.

I should also make the point that, in this context at least, you aren't special. It's tempting to believe that the correct number of notifications for your organization is a subjective thing and that you don't need to worry too much because your recipients are savvy. Let me be clear—I don't care if you're surrounded by the floating disembodied brains of particle physicists where you work inside the singularity beneath the LHC, or by coffee machines harboring nascent AIs over at JPL, they will hate you if you send them too many notifications, and "too many" is an integer that can be derived by a formula that returns the absolute number of notifications you should be sending given the number of hosts and recipients in your environment. My point is, this is a universal truth of human nature, and you NEED to worry about it; savvy is NOT an input variable. I'd give you the formula, but I haven't had a chance to work it out yet (when I do, I'm writing a LISA paper about it, though).

3. Eliminate Email Dependencies

Once you've minimized the number of notifications you send, you should proceed to make darn sure the ones you're sending are getting delivered. A few months back I wrote an entire article [3] on the subject of this tip, so I'll spare you the rant and summarize by saying that my faith in email is waning. Instead, I recommend text via SMS, voice notification via Asterisk, or a combination of the two. Email-SMS gateways are OK, a real SMS modem attached to the system is better, real SMS with voice backup is best. My article walks you through the configuration of all of that.

Even if you do stick with email, an out-of-band backup is a great way to make the monitoring system resilient against network outages, which is helpful for the same credibility-related reasons listed in tip #2.

4. Monitor the Monitoring System

This one is self-explanatory. Few of us are good at introspection, and Nagios being no exception, it's wise to have at least a couple of heartbeat scripts somewhere off the monitoring system to make sure the box and daemon are running. In the past, we had separate boxes for monitoring and logging, with the logging box watching Nagios. These days I have multiple special-purpose Nagios systems watching each other.

5. Have a Naming Convention

I cannot stress enough how important it is to have a predictable naming convention for the hosts and services in your Nagios implementation. The CPU_LOAD service should be called the CPU_LOAD service everywhere it is consumed, measured, reported on, and referred to throughout your environment. It should transcend disparate monitoring systems, executive reports, event-correlated databases, and Web front-ends. The host called fooServer02.hq.com should be referred to everywhere as exactly fooServer02.hq.com, not fooServer2, or fooserver02.hq, or any other derivation thereof. "www.foo.com" should never be referred to as "fooweb" or "the foosite" or anything other than "www.foo.com."

Effective monitoring systems grow. They quickly become relied upon to prove SLA compliance and provide re-purposed availability information to executives, customers, and other technical staff members. When this happens, programs will be written to query and move data around. As things get bigger, ancillary systems will come in—RRDtool, Cacti, etc. If you aren't anal-retentive about names from the get-go, then things will quickly devolve into a kludgey mess. The RRDtool database referring to fooServer02.hq.com will not match the name in Nagios, or people will write scripts assuming different names. Data tables and reports will be empty for some systems but not others, and it will appear to be the monitoring systems' fault for not collecting data.

Worse, it's nearly impossible to fix these sorts of problems without a proper and agreed-upon naming convention in place. Every new system, service, or change introduces the possibility of another statically coded name exception in one or more of the four thousand tiny data-mover scripts. Credibility is quickly lost in an environment like this.

Your naming convention should be so pervasive that literal service names start leaking into human vernacular. When people start saying things like "CPU underscore Load" in meetings, you're on the right track.

6. Aggressively Collect Performance Data

You may have wondered in tip #2 why you would want to monitor lots of services if you weren't going to notify on them. Performance data is the answer. There is no good reason why you shouldn't collect performance data on every service that you poll. In Nagios even plugins that don't officially return performance data via the pipe syntax can be parsed directly for performance data. Tools are available to completely automate the detection of new services on new hosts and to create and maintain round robin databases of performance data for them. Even if you don't have the means or the inclination to display performance data, you should be collecting it in case you ever want to.

For years I have used the combination of NagiosGraph [4] and Drraw [5] to glue Nagios to RRDtool. NagiosGraph does an awesome job of completely automating the task of getting data out of Nagios and into round robin databases. It detects new services and hosts using regular expressions, and creates new RRDs as necessary. After the initial setup, you don't need to do a thing, and you'll have performance data for every service on every host you monitor. Drraw is a super-simple CGI-based Web app that takes a directory of RRDs and gives you interfaces to draw anything from individual graphs to dashboards. It is the most flexible interface for drawing graphs from RRDs I've used. It makes it easy for me to quickly draw a graph that cross-references data from all sorts of hosts in different locations, and I don't feel I need to keep the graph around. Usually I just draw it to get a question answered and never save it at all. This sort of quick, informal graphical troubleshooting has become an important tool for me, and I'd be an unhappy sysadmin without it. I highly recommend both of these tools.

7. Implement Purpose-ful Nagios Systems

Finally, if you have the resources, it's a great idea to consider running disparate Nagios daemons for different purposes. For example, we run two different kinds of Nagios daemons where I work, "internal" and "external." The internal Nagios hosts sit in the production environment with the production systems and query the NRPE-type services: CPU, memory, swap, disk space, ps lists, and the like. The external Nagios daemons sit on the public Internet and act like customers, logging into the public Web sites, authenticating, clicking around, doing things that humans do.

We don't bother rolling up alerts, preferring instead to let Nagios hosts individually contact us about things they think are wrong. In this way we get a much better feel for not only how reliable our services are but how reliable our Nagios hosts are, and we gain a measure of clarity about a given problem based on which hosts are complaining and about what things. A box on an XO link in California complaining about a problem that hosts in Texas and Pennsylvania don't see could imply an upstream network outage, for example. This also has a tendency to keep the server configuration simple and transparent.

That about wraps it up. I hope these weren't overly obvious and that you perhaps found something that might help you out in the future. There are a lot of places to easily go wrong implementing monitoring systems, so oftentimes it's the human equation that makes a huge difference between good systems and bad ones. A huge difference, I dare say, between two sides of a very fine line.

Take it easy.

REFERENCES

[1] djbdns: http://cr.yp.to/djbdns.html.

[2] Holt-Winters and exponential smoothing: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc437.htm.

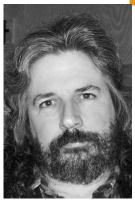
[3] Dave Josephsen, "iVoyeur: Message in a Bottle—Replacing Email Warnings with SMS": http://www.usenix.org/publications/login/2009-02/pdfs/josephsen.pdf.

[4] NagiosGraph: http://sourceforge.net/projects/nagiosgraph/develop.

[5] Drraw: http://web.taranis.org/drraw/.

ROBERT G. FERRELL

/dev/random: a realist's glossary of terms widely employed in the information security arena



Robert G. Ferrell is an information security geek biding his time until that genius grant finally comes through.

rgferrell@gmail.com

Accountability: the principle that actions taken on a system can be traced to a specific user who is not under any circumstances you.

Accreditation: a decision taken by a senior management official to allow an information system to operate securely so long as it doesn't negatively impact the budget.

Advanced Encryption Standard: the protocol employed to produce most legislation and contractual documents.

Antivirus: a signature-based software product which sometimes prevents malicious code that the black hat community has long since stopped deploying from infecting an information system.

Botnet: a dynamically distributed sensor array for monitoring the aggregate online user IQ in real time.

Certification: the miraculous process of converting money into expertise by filling in scan sheets.

Compensating Controls: those wholly ineffective measures stipulated by management as a cost-saving substitute for the recommended security controls.

Compromise: the point at which most senior managers realize those memos they've been getting from the information security staff the past few weeks/months/years were not just instances of employee whining, after all.

Cross-Site Scripting (XSS): an undocumented feature of most Web browsers.

Cryptography: the process by which the real meaning of content is obfuscated. Examples include EULAs, legislation, and telephone bills.

Disaster Recovery Plan: a detailed strategy for dealing with the impact of poor executive decision-making.

Distributed Denial of Service: technical name for the Worldwide Web.

Hacking: the process of employing a computer system to some significant fraction of its full potential.

Honeypot: a site where your actions and habits are closely scrutinized; i.e., virtually any commercial site on the Web today.

Incident: something that happens to other people.

Incident Handling: the policies and procedures in place to deal with user behavior.

Information Security: a dangerous, wholly imaginary state achieved by "experts" through a mixture of placebos, false promises, and outright fabrication. Alternate meaning: the condition achieved when all sources of power for the system have been disabled.

Information System Security Officer: a hapless individual charged with maintaining a strong information assurance posture without sufficient resources or management support.

Insider Threat: the elephant in the computer room.

IT Security Investment: a negatively asymptotic value tending toward zero.

Kerberos: a system for secure authentication so long as no one else is listening.

Macro Virus: a feature of most word processing and spreadsheet applications.

Man-in-the-Middle Attack: a bucket brigade where one of the participants substitutes kerosene for water.

Memorandum of Understanding: a mostly incomprehensible document wherein both sides agree not to take responsibility for the security of a shared information system.

Mission Critical: any software or hardware that, if it fails to function properly, may jeopardize the bonus of an executive or senior manager.

Password: a means of identification and authentication that experiences a dramatic loss of efficacy once it passes a length of about eight characters.

Patching: the penultimate phase of the commercial software development lifecycle, immediately preceding cessation of vendor support to the end user.

Phishing: a means for collecting sensitive personal information over the Web. See also: *eCommerce*.

Plan of Action and Milestones: a document detailing the tasks that need to be accomplished and estimating a completion date for each, created to postpone as long as possible actually doing any work to achieve those goals.

Port Scan: a diversionary tactic designed to keep intrusion detection systems and security administrators occupied while the real damage is being done elsewhere.

Residual Risk: that which remains after the IT security budget is exhausted, usually approximately ten working days into the fiscal year.

Responsible Individual: the person whose fault it is when something bad happens. Also known as "that other guy."

Risk Management: the process by which stakeholders and executive leadership are made to feel all warm and fuzzy about the organization's security posture by the use of empty assertions, inane media-created buzzwords, and meaningless jargon.

Secure Socket Layer: a Web-based protocol employed to give users the illusion that their transactions are secure by displaying a little padlock in the status bar. The fact that padlocks can be broken with one swing of a sledgehammer is generally ignored.

Social Engineering: the precept that people will tell you anything you want to know if you ask nicely enough.

Spyware: any software that has been downloaded from the Internet, either with or without the user's conscious participation.

SQL Injection: a clever programming trick employed primarily by obscure commercial sites to augment their media footprint.

System Administrator: see *Scapegoat*.

Threat Assessment: a careful perusal of the employee directory.

Training: the egregiously mistaken belief that “boot camps” are all it takes to prepare someone for the real world.

Trojan: malware that infects your computer surreptitiously if you’re not practicing safe surfing. See also: *Irony*.

User:

1) the principal threat to any information system.

2) the justification for existence of any information system.

Virus: a nasty piece of malicious code that wriggles its way into your machine and disrupts its functioning to the point of non-usability. See also: *Operating System*.

book reviews

ELIZABETH ZWICKY,
WITH JEFF BERG, BRANDON CHING,
AND RIK FARROW

DATA CRUNCHING: SOLVE EVERYDAY PROBLEMS USING JAVA, PYTHON, AND MORE

Greg Wilson

Pragmatic Bookshelf, 2005. 188 pp.
ISBN 0-9745140-7-1

Last issue, I reviewed (and loved) *Automating System Administration with Perl*. Let's suppose you want something similar, but you don't like Perl. This is the book for you. What it means by "Java, Python, and more" is basically "Java, Python, a little bit of Ruby, the occasional mention of C." It only brings up Perl to sneer at it.

Data Crunching doesn't cover the same range as *Automating System Administration with Perl*. Obviously, some of this is because it's not about system administration, but the result is that *Data Crunching* focuses on data sitting still, while *Automating System Administration with Perl* also covers notification and data gathering techniques. Thus, *Data Crunching* and *Automating System Administration with Perl* both give you basics of XML, XSLT, and SQL, but *Automating System Administration with Perl* adds DNS, SNMP, SMTP, and log files, while *Data Crunching* provides more coverage of regular expressions and of programming technique.

Data Crunching gives you the basics you need to know in order to beat a data problem to death with programming with an appropriate level of elegance, and when you should choose to use a different solution (either just doing it by hand or doing a proper job of programming).

STAND BACK AND DELIVER

Pollyanna Pixton, Niel Nickolaisen, Todd Little, and Kent McDonald

Addison Wesley, 2009. 152 pp.
ISBN 978-0-321-57288-2

I love management techniques that involve low intervention, so I was predisposed to like this book, and I liked it just as much as I hoped. But as it turns out, my absolute favorite part is not about hands-off, high participation management; it's about convincing people NOT to do things. Lots and lots of books will tell you about ways of convincing people to do things, or helping them choose between options, but very few people will tell you what to do if, as far as you can tell, your entire project is obsessed with choosing between and rank ordering things they just shouldn't be doing at all. It can be agonizingly difficult to convince people that the problem is that they are doing a perfectly competent, well-managed, on-time job—of the Wrong Thing. *Stand Back and Deliver* will not fix all of these situations, but it provides tools for opening up the conversation to the idea of doing some things just well enough, and the idea that you might want to totally rethink the goals of your project.

This is a book for people who already have a grasp of the basics of managing projects and need some new techniques; it's not a full toolbox, it's a set of interesting, specialized tools you may not have seen before. But they're very useful tools that you won't find elsewhere. If you need some new ways of looking at large projects, check it out.

GRACE HOPPER AND THE INVENTION OF THE INFORMATION AGE

Kurt W. Beyer

MIT Press, 2009. 380 pp.
ISBN 978-0-262-01310-9

This is a fascinating biography of Grace Hopper. It's an academic biography, which means that it hews pretty carefully to knowable facts, without dramatizing (in fact, sometimes it seems to be burying them a bit), but the facts are riveting ones if you're at all interested in the history of computing. Sometimes you marvel at how much things have changed, and sometimes you marvel at how little they've changed. The startup experience and the professional group formation experience are essentially unchanged during the time when the computers themselves and the process of programming them have both changed beyond all recognition.

This is mostly about the history of computing, with some discussion of Hopper's position in the history of women in technology. That doesn't appear to be the author's main interest, but you can't really avoid it when you're talking about the most famous woman in the development of modern computing. There's a very delicate balancing act to be done here. Grace Hopper is sometimes thought of as evidence that women have always been accepted in computing, and sometimes held up as an exceptional person who succeeded despite being female and is not representative. As always, there's truth on both sides. The book does a nice job of trying to provide context for Hopper's achievements.

One warning; the chapter labeled 1 is effectively a prologue. It's more academic and less interesting than the rest of the book, and includes some meta-argumentation about academic biographies that will only be absorbing to those interested in academic biography as a genre rather than in Grace Hopper and the history of computing. I wish I'd started with Chapter 2, which is where the actual story starts.

ALGORITHMS FOR VISUAL DESIGN USING THE PROCESSING LANGUAGE

Kostas Terzidis

Wiley, 2009. 337 pp.
ISBN 978-0-470-37548-8

Theoretically, this book is aimed primarily at people with design and architecture backgrounds who are learning programming from the ground up. This is one of those rare cases where the title does a better job of selecting an audience than the introduction does; don't try this book on somebody who can't already program in some language, because its introduction to programming takes about 30 pages, followed by exercises such as "Write the shortest possible procedural code that can generate the following number pattern using only standard arithmetic operations." You need to be reasonably literate in programming, mathematics, and visual design: not an expert, but comfortable looking at equations, or code samples, or pictures of patterns.

If you're in that audience (or, I suppose, if you are strongly motivated and like a challenge), this book does a nice job of introducing Processing, plus some native Java constructs you might need, and the sorts of algorithms you are likely to want to use to do visual stuff. It covers some stuff that *Processing* (reviewed earlier this year) does not, mostly by going at breakneck speed and including less art.

I found this book useful and enjoyed it, but it has a number of drawbacks. To start with, don't read the introductions to the chapters. Introducing chapters is always tricky, and as an author I sympathize with the struggle to work your way into the meat of the chapter. But if you're looking for advice on writing out files, the following is not going to entice you into the chapter:

> Memory is the mental faculty of retaining and recalling past experience; it is the act of remembering or recollecting.

You'll also probably want another book on Processing, because the author is fond of doing things the hard way while failing to fully explain the easy way. It's nice to know the math behind Bézier curves, but actually implementing them from scratch is not the right answer in a language which has curve primitives built in. Once you understand what's going on, use the primitives. Similarly, I'm glad to know how to do native Java file writes from Processing, but it would have been friendlier and more useful to start by using the handy built-ins. It would also be friendlier to clearly distinguish between Processing constructs and Java constructs, at least so that students can easily figure out what manual they ought to be trying to look things up in, and they will have some idea of what Java constructs will work.

MALWARE FORENSICS: INVESTIGATING AND ANALYZING MALICIOUS CODE

Cameron Malin, Eoghan Casey, and James Aquilina

Syngress, 2008. 592 pp.
ISBN 978-1597492683

REVIEWED BY JEFF BERG

A book running over five hundred pages might seem a bit cumbersome, but *Malware Forensics: Investigating and Analyzing Malicious Code* provides a jump start into malicious code incident response and the issues that encompass it. Even those involved with malicious code research on a daily basis will find this book to be a good tune-up on the methodology, tools, and techniques implemented for effective incident response. Covering everything from retrieving potential artifacts from physical memory and volatile information to minimizing changes to a system, backing up hard drives for further analysis, and the legal ramifications of investigations, *Malware Forensics* discusses all the major topics and then some.

One of the underlying concepts that echoes throughout the book is evidence dynamics—a term that has been coined to reflect influences that will change, relocate, obscure, or damage evidence. This is most easily applied to physical criminal cases in which a first responder may "disturb the scene" where a victim is found, but translates to malicious code incident response in the way a host or victim is influenced as an investigation begins. Evidence dynamics is an important concept regardless of the specific profession a researcher embarks upon, because it is impossible to know when data collected will have to be used as evidence. The authors do a great job of addressing this topic throughout the entire book.

One excellent feature of this book is the practical case scenarios that are found in each chapter and, in some cases, carried on throughout the book. The scenarios enable the reader to translate the concepts into the real world. The authors also provide helpful analysis tips, such as processing suspect files and conducting analyses in isolated environments, a tip that seems to be common sense to most veteran researchers but might not be apparent to the newbie. Another cool aspect of this book is that the authors constantly suggest tools that could be helpful, along with pointers about how to obtain the tools. However, a good part of the book's value lies not so much in what is included but in what is excluded. The authors defer to other sources on general knowledge-base topics such as network traffic analysis aiding in an investigation, as well as background knowledge of ext2 and ext3 file systems. This is as it should be, but the reader should be aware of the expected prerequisite knowledge and that he or she may need to crack a second book to get up to speed for the provided discussion.

Malware Forensics is broken into ten chapters, some of which cover the same general concepts but concentrate specifically on Windows or Linux. For example, Chapters 1 & 2 cover "Volatile Data Collection & Examination on a Live . . ." Windows or Linux system, respectively. The two chapters explain the methodology, tools, and techniques involved with data collection during the initial portion of a response (e.g., documenting network connections, logged-on users, open ports, process information, etc.), all while keeping evidence dynamics in mind, documenting steps and forensic preservation for deeper analysis. Chapters 4 & 5 discuss the "post-mortem" phase of the analysis—gleaning information from the preserved copies of data and introducing a methodology that is repeatable and used in either a random infection or a test infection, where a system is purposely infected to better understand the piece of code. Chapters 7 & 8 cover "File Identification & Profiling . . ." The authors discuss a methodology, fairly similar across Windows and Linux, for pulling malicious executables or files related to them off the hard drive and "profiling" them—i.e., getting the hash, determining file type, scanning with a malicious code scanner, and running strings for references or keywords to assist in classifying the file and, ultimately, the code. Chapters 9 & 10, "Analysis of a Suspect Program . . . ," tie a lot of the concepts covered throughout the book together into a chapter-long case study of a potential malicious program.

The only two chapters that do not focus on Windows or Linux specifically are 3 and 6. Chapter 3 discusses the methodology, techniques, and different tools necessary for acquiring and analyzing memory for malicious code evidence on both Windows and Linux systems. Chapter 6 is dedicated to the legal aspects of the field, covering issues such as who has jurisdictional authority to conduct an investigation, ensuring that an investigation is performed such that the evidence is admissible, and providing basic guidance for instances where evidence may exist outside jurisdictional authority and how to obtain it. Though the authors stress the need to consult with legal teams, they provide a good base of issues to be aware of.

Malware Forensics will dive as deep as analyzing a suspect program or process in a disassembler to understand what a program is doing, but it doesn't require a reverse engineer's background to gain a good amount of value from reading it. However, the reader should be prepared to spend time with the tools and techniques discussed. If you are anything like me, you'll benefit much more by doing so. And if you are truly worried about the length, practicing with the tools after each chapter will break up any monotony of reading.

SEXY WEB DESIGN: CREATING INTERFACES THAT WORK

Elliot Jay Stocks

SitePoint, 2009. 172 pp.
ISBN 978-0980455236

REVIEWED BY BRANDON CHING

I do not have a single creative bone in my body! OK, that's probably an exaggeration, but when it comes to designing an innovative, attractive, and usable Web site, I definitely could use a helping hand. As a Web developer, I am generally responsible for the data in our sites rather than the look and feel; that's the UI team's domain!

However, not all developers have access to professional UI resources and, depending on the situation, many of us often wear a number of different hats.

As such, *Sexy Web Design* is a book that seems made for folks like me who know a little something about the basics of Web design, but are nowhere near creative experts.

The primary focus of *Sexy Web Design* is on the process and fundamentals of professional design rather than specific HTML and CSS techniques. As such, there is a heavy focus on planning and organizing a Web site and little coverage of actual coding. In fact, there is little to no CSS in the book. You may now be asking yourself, what use is a design book without any code samples? In short, quite a bit.

The author takes you through all phases of planning a site design, from research and customer requirements to site mapping, wire framing, usability, composition, navigation, and his keen emphasis on aesthetics. The text is an ideal companion for designers and developers who are new to interacting directly with customers, as Stocks covers the professional give-and-take process of designer/customer interactions. While coverage of each topic is rather short, it is to the point and seems to cover the requisite ground for a basic introduction.

There are many good visual examples of the principles Stocks is introducing, and he does point the reader to more detailed texts and Web sites for further reading. The text also provides a number of general considerations that people not trained in design could be unfamiliar with, including composition, mood, contrast, volume and depth, typography, and textures.

Overall, the book was very readable and approachable. I would recommend the book to experienced Web developers who don't dabble much in site design or customer interactions (but know the technical details of HTML and CSS) and for those looking for a general introduction to the planning and creative considerations of designing a Web site from the ground up.

I would say that the biggest disappointment of the book was its brevity. Stocks covers a lot of ground in only 172 pages, and with a lot of that real estate used up with visual aids and examples, it doesn't leave much for explanation. However, in combination with other more detailed and technical resources, it does creatively address the often neglected planning, usability, and customer interaction aspects of a bottom-up site design, and for that, I feel that it is a book worthy of consideration.

Web 2.0 Architectures: What Entrepreneurs and Information Architects Need to Know

James Governor, Dion Hinchcliffe, and Duane Nickull

O'Reilly, 2009. 271 pp.
ISBN 9780596514433

REVIEWED BY BRANDON CHING

There are many Web 2.0 books out there these days, but most seem to address the Web 2.0 concept in a business or abstract sense. *Web 2.0 Architectures* by Governor et al. is written for "in the trenches" IT professionals who are charged with the creation and execution of next-generation Web platforms.

Web 2.0 Architectures takes an in-depth look at system architectures from a practical and theoretical perspective relative to Web 2.0 strategies. The book can be seen as being organized into three general areas: introduction and models, architecture patterns, and building for the future.

The first six chapters take you deep into the technical details of Web 2.0 architecture, network, and design principles. The real heavyweight of this section is Chapter 3, where the authors compare and contrast the Web 1.0 and Web 2.0 design patterns of some major companies and concepts in tech, including Akamai/BitTorrent, MP3.com/Napster, and CMSes/wikis. The comparisons are detailed and informative, outlining the major shifts in thinking and design between the approaches. There are plenty of diagrams, flow charts, and network models to emphasize the message. After this section, the reader will have a solid foundation for the more theoretical concepts to come.

Chapter 7, comprising about half the book's page count, is where the authors deliver the main content: detailed coverage of Web 2.0 architecture patterns. The authors discuss twelve patterns in detail, including an explanation of the business problem, solution, behavior, implementation, consequences, and more. Again, there is no shortage of charts and diagrams to help impress upon the reader the lessons being taught, and superb lessons they are indeed.

Finally, Chapter 8 attempts to tie up the loose ends, describing where Web 2.0 is going and identifying offshoots of next-generation platforms such as Advertising 2.0 and Government 2.0. This chapter was not very strong, and I didn't find much useful information that was relevant to the rest of the book.

Overall, *Web 2.0 Architectures* is well worth a read for its comparative analysis of Web 2.0 models and its detailed explanation of Web 2.0 patterns. This book is ideal for senior-level information architects and technically minded entrepreneurs who are looking to build a Web 2.0 system architecture from the ground up. The language and topics covered are definitely on the technical side, so this book may be out reach for some, but the writing is clear and professional. If you (or your business) are looking to adopt a more advanced and focused architecture but need a bit of conceptual and theoretical guidance, this book is definitely where you should start.

IWORK '09: THE MISSING MANUAL

Josh Clark

O'Reilly Media/Pogue Press, 2009. 896 pp.
ISBN 978-0-596-15758-6; eBook 978-0-596-80341-4

REVIEWED BY RIK FARROW

I wanted to try an eBook, and I also needed to know more about Keynote, the presentation software that Al Gore made famous. I had used Keynote to make several presentations, and can say that it is *mostly* intuitive and easy to use. But there were certainly some aspects that just had me baffled.

iWork covers Pages, Keynote, and Numbers, the Word, PowerPoint and Excel-like programs from Apple. I focused on the Keynote section, as that is what I was using, and quickly found that I liked reading the book. Clark starts the Keynote section with advice about creating presentations that I actually found useful. Instead of skipping over what would be filler in another book, I read most of this chapter of good advice.

Getting into the nitty-gritty, I was able to quickly find answers to things that you might think would be easy to do, like printing a two-up (two slides per page) version of your slides. I had puzzled over the Print dialogue several times, but Clark explains that using a menu and selecting the Layout tab makes it easy to do this. I had never used animation, and again Clark made this easy to understand.

iWork does have features that overlap between applications, such as fonts and drawing tools, so the Keynote section does not stand alone. But that is not a failure of the book, just me wanting to get answers in a hurry.

I am still not comfortable with eBooks: laptops do not work that well for reading in a comfy chair or in bed, the screen format is not portrait but landscape, the laptop will start to burn your skin, among other interesting failings. Owning a Kindle, which implies that I can no longer lend a book to a friend without lending my entire library, is not the model for me. I did try borrowing the large LCD display known as a TV these days, but again the aspect is landscape, not portrait. I found that the eBook was good as a reference and not so good for sitting down and reading—nowhere near as convenient to use as a plain old paper book.

I can recommend *iWork '09*, the book, as a useful companion to the Apple apps, and well worth the price for the time saved searching for answers to simple questions, as well as for Clark's useful advice.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

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

ACCESS TO ;LOGIN: online from October 1997 to this month: www.usenix.org/publications/login/.

DISCOUNTS on registration fees for all USENIX conferences.

SPECIAL DISCOUNTS on a variety of products, books, software, and periodicals: www.usenix.org/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

2010 ELECTION FOR THE USENIX BOARD OF DIRECTORS

Ellie Young, Executive Director

The biennial election for officers and directors of the Association will be held in the spring of 2010. A report from the Nominating Committee will be emailed to USENIX members and posted to the USENIX Web site in December 2009 and will be published in the February 2010 issue of *;login:*.

Nominations from the membership are open until January 6, 2010. To nominate an individual, send a written statement of nomination signed by at least five (5) members in good standing, or five separately signed nominations for the same person, to the Executive Director at the Association offices, to be received by noon PST, January 6, 2010. Please prepare a plain-text Candidate's Statement and send both the statement and a 600 dpi photograph to jel@usenix.org, to be included in the ballots.

Ballots will be mailed to all paid-up members in early February 2010. Ballots must be received in the USENIX offices by March 17, 2010. 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 meeting following the election, or on July 1, 2010, whichever comes earlier.

USACO TEAMS SHINE

Dr. Rob Kolstad, Head Coach, USA Computing Olympiad

USENIX-sponsored USA Computing Olympiad teams won the team championship at the Central European Olympiad on Informatics and placed second at the world championship IOI in Plovdiv, Bulgaria.

After a very competitive training camp held for the USA's top 15 competitors during the first week of June at the University of Wisconsin—Parkside, extra funding enabled our traveling teams to participate in two international championships.

Coaches Rob Kolstad and Jacob Steinhardt traveled to Târgu-Mureş, Romania, for the July 8–14 Central European Olympiad on Informatics. The CEOI is occasionally persuaded to invite the USA as a guest team; we think of ourselves as sort of the western region of Central Europe. A total of 57 competitors from 11 countries converged on Romania for the intense competition.

This Olympiad was just 100km east of the location of our previous CEOI in Romania in Cluj, and still in the heart of Transylvania. Dracula's influence was felt throughout our stay, especially when visiting tourist spots.

Compared to the 300+ competitors at the international championships (IOI), these competitions are relatively intimate and much more challenging for medals. Because of the limited number of medals available (only 1/12 of participants earn Gold medals; 2/12 for Silver; 3/12 for Bronze) and about the toughest competition in the world (except for China and a few individuals in random countries), the most prestigious medals are very difficult to win. The intimacy does enable the students to get to know each other much better, especially since the cultures are dramatically less diverse and easier to absorb.

The CEOI results were fabulous! Four USA students competed at the CEOI. IOI alternate Michael Cohen had a great finish in his first international outing, and three of our team won medals, as well as placing in the top 10:

- Gold Medal, #1, Neal Wu, from Baton Rouge, LA
- Silver Medal, #6, Michael Cohen, from Chevy Chase, MD
- Silver Medal, #8, Brian Hamrick, from Annandale, VA
- #36, Travis Hance, from West Chester, OH

Neal's 400 points were 60 more than the second place winner's (a stunning victory margin), and he was one of only three students earning above 300.

Just a month later, Coaches Rob Kolstad, Brian Dean from Clemson, and Richard Peng from the University of Waterloo escorted Neal, Brian, Travis, and Wenyu Cao from Belle Mead, NJ, to the International Olympiad on Informatics in beautiful Plovdiv, Bulgaria. (Wenyu had been unable to attend the CEOI, because it conflicted with the International Math Olympiad where he was representing the USA in an allied competition.)

The two-hour bus trip from the airport in Bulgaria's capital of Sofia wound through oak- and fir-covered rolling hills. The decreasing angle of the sun really brought out the natural beauty of central Bulgaria (just the first of many aspects of the trip that exceeded any rational expectations).

The coaches' hotel, seven minutes away from the students' hotel, sported rooms that were nicer than any non-suite I've ever stayed in. A relatively large anteroom led to the main bedroom with its living area slightly separated from the sleeping section. Windows looked out over the four swimming pools, with the city lights glimmering in the distance. Many meals were served poolside from a buffet with more than 100 items, almost all of which I found edible! What a treat for this year's IOI accommodations.

Like the CEOI, two five-hour competitions (each with one easy and three extremely challenging tasks) were interspersed with tours of the re-

CEIO team, left to right: Michael Cohen, Travis Hance, Brian Hamrick, Neal Wu, Coach Jacob Steinhardt

IOI team, left to right: Travis Hance, Wenyu Cao, Brian Hamrick, Neal Wu gion and of the city. The "old city" adjoined the students' hotel, so sightseeing was without stress.

Our students performed extremely well in the event:

- Gold Medal, #7, Neal Wu, 665 points
- Gold Medal, #15, Brian Hamrick, 642 points
- Silver Medal, #28, Wenyu Cao, 613 points
- Silver Medal, #40, Travis Hance, 566 points

Wenyu was just three points away from a gold medal. The USA's total points ranked second among all countries, exceeded only by powerhouse China.

The awards ceremony was held in a 2,000-year-old Roman amphitheater. Not only were our competitors honored by medals, but Coach Kolstad was awarded the Distinguished Service Medal for contributions to the IOI and training of competitors from countries around the world.

All in all, a fabulous end to a super season for USACO.

USENIX is a principal sponsor of USACO, along with Booz Allen Hamilton and IBM.

The USA Computing Olympiad continues to hold monthly competitions for three divisions of competitive programmers. This year's contests have each drawn 1,000 or more competitors from more than 60 countries. Thanks to USENIX and its membership for continuing sponsorship of this great program.

Statement of Ownership, Management, and Circulation, 10/1/09

Title: ;login: Pub. No. 0008-334. Frequency: Bimonthly. Subscription price $125.
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: Jane-Ellen Long, 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 (Oct. 2009) published nearest to filing date of 10/1/09*
A. Total number of copies	5745	5508
B. Paid circulation		
Outside-county mail subscriptions	3420	3181
In-county subscriptions	0	0
Other non-USPS parcel distribution	1684	1698
Other classes	0	0
C. Total paid distribution	5104	4879
D. Free distribution by mail		
Outside-county	0	0
In-county	0	0
Other classes mailed through the USPS	76	73
E. Free distribution outside the mail	341	300
F. Total free distribution	417	373
G. Total distribution	5521	5252
H. Copies not distributed	224	256
I. Total	5745	5508
Percent Paid and/or Requested Circulation	89%	89%

I certify that the statements made by me above are correct and complete.
Jane-Ellen Long, Managing Editor

conference reports

THANKS TO OUR SUMMARIZERS

18th USENIX Security Symposium

Montreal, Canada
August 10–14, 2009

OPENING REMARKS AND AWARDS

Summarized by Rik Farrow

Fabian Monrose began with the statistics: 176 papers were submitted to the symposium. One was withdrawn and three rejected for double or triple submissions to conferences (resulting in the papers being automatically rejected from all the conferences). Three or more people reviewed each paper, with Monrose reading the vast majority of these papers. Many submissions were being edited right up to the deadline. By the time of the PC meeting, there were only 62 papers left to discuss. All PC members attended the meeting in Chapel Hill, and Monrose said there were some real battles there (beside the NCAA tournament). By the end of the meeting, 26 papers were accepted.

There were 84 applications for student grant support; the USENIX Board provided $50,000. The two Outstanding Student Paper awards were "Compromising Electromagnetic Emanations of Wired and Wireless Keyboards" (Martin Vuagnoux and Sylvain Pasini, LASEC/EPFL) and "Vanish: Increasing Data Privacy with Self-Destructing Data" (Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy, University of Washington).

KEYNOTE ADDRESS

- ***Android: Securing a Mobile Platform from the Ground Up***
  Rich Cannings, Android Security Leader, Google

Summarized by Italo Dacosta (idacosta@gatech.edu)

With the increase in the adoption of smartphones as well as in our reliance on these devices, they will undoubtedly become the next target of cybercriminals. This makes the security of the mobile operating systems an area of critical importance. Rich Cannings described the main features of Android, Google's open source mobile OS, and pointed out that its openness differentiates Android from other popular mobile OSes. In Android, any user can develop applications, because there is no centralized software signing authority, but this openness also makes Android more vulnerable to malicious software. Being aware of this risk, Android developers follow a security strategy based on four components—prevent, minimize, detect, and react—to protect Android's core components, applications, and user data.

To prevent possible attacks based on the exploitation of unknown vulnerabilities, Android has partnered with security experts to target high risk areas such as remote attacks and vulnerabilities in media codecs. Being an open source OS, Android does not rely on obscurity techniques for its security. In addition, well-known security mechanisms such as stack overflow protection (ProPolice) and heap protection (dlmalloc) have been implemented. Address Space Layout Randomization (ASLR) has not been implemented yet due to some platform constraints but is expected to be added in the future.

Cannings said that not only is it unfeasible to prevent all the possible security vulnerabilities in Android, but attackers do not always even need vulnerabilities to compromise an OS; social engineering techniques and bugs can also be used to install malware. Therefore, it is important to minimize the impact of compromised applications. For this, Android tries to extend the Web security model to the OS, using an application sandbox model for separation of privileges (each application runs with its own UID and virtual machine). Applications are locked down to their minimal functionality, and permissions are required to grant more access to resources, a whitelist model. Users decide to give permissions or not to the applications when they are installed. A challenge in this area is to determine the right number of permissions that should be asked of the user (granularity) because too many questions could cause the user to ignore this mechanism. In addition, media codec libraries are given lesser privileges than in other OSes, given the long history of vulnerabilities in media codecs.

To detect attacks, Android uses activities such as developer education, code audits, fuzzing tests, and honeypots. Because Android is an open source OS, anybody can detect and report security problems: users, developers, security researchers, etc. External reports from members of the security community have helped Android's developers fix several security problems. Also, users are encouraged to report suspicious applications in the Android Market. Reported applications are analyzed by Android personnel and removed from the Market if they are considered malicious.

Android relies on auto-updates to distributed security patches to fix critical security vulnerabilities. Android uses an over-the-air update system where user interaction is optional and no additional cables or computers are required, resulting in a high update rate. However, the main challenge to apply security updates is the testing of the updates and the coordination with different mobile network providers. For mobile carriers, updates are a concern because they could affect the availability of a great number of devices, resulting in financial and customer service problems. Therefore, before being released, security updates should be carefully tested and approved by each mobile carrier, but this process can delay the release of the update considerably.

During the Q&A, Rik Farrow asked about the prevention of privilege escalation attacks. Cannings answered that one way to mitigate this type of attack is to reduce the number of processes running with root privileges. In Android, only ping and zygote (the application launcher) run with root privileges. Regarding the support of security hardware mechanisms, Cannings commented that they are evaluating the use of the execution prevention bit and other hardware mechanisms. How many of Android's 5 million lines of code were written in type-safe language? Most of the Android code is written in Java, not only for security but also for compatibility purposes. Finally, Gary McGraw asked what percentage of the vulnerabilities discovered in Android were discovered externally versus internally. The number of vulnerabilities discovered internally was several orders of magnitude greater than those discovered externally.

ATTACKS ON PRIVACY

Summarized by Shane Clark (ssclark@cs.umass.edu)

- ***Compromising Electromagnetic Emanations of Wired and Wireless Keyboards***
  Martin Vuagnoux and Sylvain Pasini, LASEC/EPFL

  Awarded Best Paper!

According to Martin Vuagnoux, the authors chose the keyboard as an attack vector because it is the first device in a system that handles sensitive data such as passwords electronically; security is not a priority in their design. They chose to examine electromagnetic emanations because many other attack vectors for I/O devices have been demonstrated in the past, such as electromagnetic leakage from displays and acoustic emanation from keyboards.

To provide background for the work, Vuagnoux next introduced radiative emanations and their capture. Radiative emanations are those requiring the source to act as an antenna. These are the emanations of interest for attacks, as others require physical contact with a wire. Radiative emanations can be further broken down into direct emanations (those caused by a keypress or other action) and indirect emanations (those from carrier signals, modulation schemes, etc). To observe these emanations, the authors chose to attempt to capture the entire spectrum of interest simultaneously in order to capture the maximum amount of information without scanning. They found that they were able to achieve this using a large conical antenna and a 5 GSa/s oscilloscope. After capturing the signals, they examined the Fourier transform of each to identify interesting characteristics.

Vuagnoux moved on from background and acquisition methodology to describe three attacks that are effective only against PS/2 keyboards. The first attack relies on the fact that PS/2 keyboards modulate a scan code onto a clock signal by pulling down a data line repeatedly. It is possible to exploit this direct emanation by observing the series of falling edges that this creates in the modulated signal, but this results in aliasing among the scan codes. The authors constructed a table of all keys and their corresponding signatures, which allowed them to reconstruct words typed based on the sequence of key presses. The second attack simply filters the same information and computes a distinct threshold in order to remove aliasing. The final PS/2 attack actually demodulates the captured signal in order to remove noise from the clock. The only USB attack relies on the use of a matrix scan loop to poll pressed keys. Which key has been pressed can be discerned from the delay associated with scan reporting. This attack is also effective against PS/2 and wireless keyboards. All of the attacks were effective in realistic environments, including through walls, though at ranges sometimes less than a few meters. Surprisingly, all attacks were much more effective when tested in an apartment building, owing to construction features such as common grounds and water pipes.

Xiaofeng Wang (IBM) asked what the implications are of multiple users typing in the same space simultaneously. Vuagnoux responded that they were able to fingerprint individual keyboard models based on clock signal differences. Perry Metzger asked a similar question, emphasizing the problem of separating users whose input is captured simultaneously, to which Vuagnoux responded that this should be possible in theory based on keyboard fingerprinting, but presented a technical problem because they were unable to actually trigger data capture accurately with current hardware. This is something that the authors are still working on. Had the authors attempted similar attacks against mice? They hadn't and, in fact, removed mice during their experiments to minimize noise. Had the authors attempted any countermeasures, such as wrapping a keyboard in shielding material? A few keyboards implement shielding based on the results of the TEMPEST program, but they cost hundreds of dollars and the authors were not able to purchase one without a military affiliation. Attempts by the authors to shield a keyboard themselves sometimes resulted in more visible emanations.

- ***Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems***
  Kehuan Zhang and XiaoFeng Wang, Indiana University, Bloomington

Kehuan Zhang reported a new shared information vulnerability present on multi-user UNIX-like systems and presented an example attack on Linux. Zhang started by introducing legitimate uses of shared information on Linux systems and procfs, the mechanism for this sharing. It is common for users on a system to run commands such as top in order to see which users are logged in to a system, what processes are running, and what resources they are consuming. This information-sharing is enabled by the process file system, procfs, which is a pseudo file system that is globally readable. It contains per-process data such as image name, starting address of the stack, and current stack pointer (ESP).

Zhang next discussed the attack, which performs keystroke inference using the information available in procfs. Before the attack can be mounted, the attacker must analyze the victim process offline and build a trace of the ESP variation in procfs as a result of user input. The attack also requires a multi-user system, the ability to execute programs, and a multicore CPU. These capabilities are necessary because the attacker must run a shadow process concurrently with the victim process in order to observe changes in procfs. The shadow process produces a partial ESP trace (as it is not possible to catch all changes reliably), which is then converted into a longest common subsequence problem in order to extract keystroke timings. The timings are then given as input to a Hidden Markov Model (HMM) to perform key inference. Multiple timing traces are produced in order to increase HMM accuracy.

Zhang concluded by presenting performance results and discussing countermeasures. He noted that the percentage of keystrokes detected decreases rapidly as CPU usage increases for some applications, but with CPU usage under 5%, all of the tested applications were vulnerable. Zhang showed server traces indicating that three test machines averaged under 4% CPU usage, to illustrate the feasibility of the attack on real-world systems. He next presented results for password inference using 50 keystroke captures. The authors' keystroke inference system was able to reduce the password search space to between 0.05% and 7.8% of the initial space. Zhang noted that a kernel patch to remove the compromising information leakage is the short-term solution, but suggested a complete evaluation of information leakage through shared information channels. Someone asked if this attack can be used to capture SSH keys, and Wang answered that it can.

- ***A Practical Congestion Attack on Tor Using Long Paths***
  Nathan S. Evans, University of Denver; Roger Dingledine, The Tor Project; Christian Grothoff, University of Denver

Nathan Evans gave a talk on a new Tor attack which allows the attacker to determine the path data travels through the network. The Tor system is the most popular free software used to achieve anonymity on the Internet. Tor uses *onion routing*, which forwards data through the network, peeling off a layer of encryption at each node. Each node in the network knows only the previous hop and the next hop. This is a key security goal for Tor, as the discovery of a complete circuit through the network makes it easier to deanonymize the originator of the traffic. Evans noted three design choices made by the Tor project that are relevant to his attack. First, no artificial delays are induced on any connection. Second, path length is set at a small finite number (3). Third, paths of arbitrary length through the network can be constructed.

Evans described the attack and countermeasures in more detail. The attacker must first operate a Tor exit node that is in use by the victim. Next, the attacker uses a malicious client to create a long loop in the network before connecting to the requested server. This allows the attacker to load the intermediate nodes as desired. Finally, the exit node injects a JavaScript ping command into the traffic that reports back to the malicious client and is used to measure the latency along the circuit as the attacker loads possible first hop routers. Based on the observed latency, the attacker can determine which node is the first hop. Since the attacker also operates the exit node, she can determine what server the victim is connecting to. Evans showed that attack runs are clearly distinguishable from normal Tor traffic in testing and that the attack is effective even against high bandwidth routers. Finally, he presented several possible countermeasures, including the prohibition of infinite path lengths, which the Tor developers have implemented.

INVITED TALK

- ***The Building Security in Maturity Model (BSIMM)***
  Gary McGraw, CTO, Cigital, Inc., and Brian Chess, Chief Scientist, Fortify Software

  Summarized by Salvatore Guarnieri (sammyg@cs.washington.edu)

The Building Security in Maturity Model (BSIMM, http://bsi-mm.com/) ranks your corporation's security practices against those of other corporations. This work differs largely from previous work in that it does not advocate security practices based on what seems like a good idea; it doesn't actually recommend anything. The model simply compares corporations' security practices. It is up to the users of the model to determine if they want to be like the organizations they are being compared to.

BSIMM is based on a study of nine large companies: Adobe, Depository Trust and Clearing Corporation (DTCC), EMC, Google, Microsoft, QUALCOMM, Wells Fargo, and two anonymous companies. BSIMM analyzed what these corporations were doing for software security and found some expected and some unexpected results. Two basic and expected findings were that security was an emergent property of the entire system and that secure software requires deep integration with the Security Development LifeCycle (SDLC).

Since BSIMM is a model that compares company practices, one would think that the companies one is compared to would be important. For example, an independent software vendor (ISV) would have different security concerns and practices from those of a financial institution. The BSIMM study actually found that this is not the case. Financial institutions and ISVs have approximately the same software security model. Additionally, the size of the companies in the study ranged from hundreds to thousands of software developers. In all the companies, the size of the Software Security Group (SSG) was 1% of the total software developers. This doesn't mean that the correct SSG size is 1% of developers, but if you like the security of these nine companies, maybe 1% is a pretty good target size for your SSG.

The model is a set of over 100 activities. You mark which activities your company does and then compare your results to the average of the nine studied companies. Each activity has a ranking associated with it that describes how easy it is to do. This ranking is also interpreted as how serious or mature a company is in a certain area of security. The end result is a simple comparison, but the speakers have developed a visualization that easily shows how one organization compares to the average. The model is available from the BSIMM Web site for free under a creative commons license.

There were a few surprising discoveries from the study, including the top 10 most unexpected results. These are all available on the Web site, but there were a few very interesting things that everybody does. First, everyone is doing code review, using tools and, most importantly, looking for ways to automate the process. Second, SSGs do architectural analysis. Architectural analysis is difficult, and product teams have a hard time doing it, so SSGs need to help out. Third, every organization has an SSG, but each one had a different way of starting its SSG.

More companies need to be studied. Nine is a good starting point, but few statistics are valid with only nine data points. They are already up to 17 companies with their current work and they keep looking to expand. As they get more companies, they can start to say more interesting things, such as comparing big companies to small companies.

MEMORY SAFETY

Summarized by Stephen McLaughlin (smclaugh@cse.psu.edu)

- ***Baggy Bounds Checking: An Efficient and Backwards-Compatible Defense against Out-of-Bounds Errors***
  Periklis Akritidis, Computer Laboratory, University of Cambridge; Manuel Costa and Miguel Castro, Microsoft Research, Cambridge; Steven Hand, Computer Laboratory, University of Cambridge

Periklis Akritidis described a technique called baggy bounds checking, which aims at increasing the efficiency of array bounds checking. Because type-unsafe languages such as C do not perform array bounds checking, previous research efforts have been made to add it to the language. Traditional backwards-compatible techniques (e.g., splay trees) require several memory accesses per check or use too much memory. To address this issue, the authors suggest allocating strategically sized buffers to make bounds checks more efficient.

The presented technique, baggy bounds checking (BBC), pads objects upon allocation to a size that is a power of two. Bounds checking is then performed, not on the object boundaries but on the allocation boundaries, which can be calculated from a pointer into the object and a single table lookup. Because BBC partitions memory into slots that are powers of two in size, the base address of an allocation can be found by clearing the lowest-order lg(size) bits of a pointer to an object, where lg is the log base two. A pointer to an object is used to index a global array that contains the log base two of the size of the containing slot, requiring only one byte to track the bounds of each allocation. Of course, this technique cannot detect memory accesses that are within an allocation but outside an object. This is not a problem, as the padded regions are cleared upon allocation to remove any sensitive data from previous allocations.

BBC is implemented as a compiler extension that works with the intermediate representation of a C program to modify memory allocation and add bounds checks. Heap allocation is modified to use buddy allocation at runtime, while globals are modified at compile time and the heap allocator is modified in the library. BBC was evaluated for memory and performance overhead using the Olden and SPECINT 2000 benchmark suites on Windows. Each benchmark was compiled using both BBC and splay tree bounds checking. Most surprisingly, BBC had a smaller average memory overhead than splay tree checking on both suites, although the splay tree did slightly better on most SPEC benchmarks. On the Olden tests, the splay tree version created 170% memory overhead, while BBC sometimes performed better than the default Windows allocator. Both of these effects are a result of the metadata overhead caused by the Olden benchmark's many small allocations. The ability of BBC to detect bounds errors was tested utilizing 18 buffer overflows from a suite of benchmarks used to test for memory errors. BBC detected 17 out of 18 errors. In the one exception, an array was allocated inside a structure. An out of bounds access to the array caused another pointer in the structure to be overwritten.

Someone asked how this scheme differed from SoftBound, which was presented at PLDI 2009. Akritidis said that SoftBound requires eight bytes to be stored for each pointer, causing high memory overheads.

- ***Dynamic Test Generation to Find Integer Bugs in x86 Binary Linux Programs***
  David Molnar, Xue Cong Li, and David A. Wagner, University of California, Berkeley

Dave Molnar presented work on generating better test cases for finding integer bugs with fuzz testing and compared it to a black box fuzz tester running in Amazon's Elastic Compute Cloud. A large number of software errors are caused by integer bugs such as over- and underflows, non-value preserving conversions, and signed and unsigned conversion errors. Typical black box fuzz testing does not deal with integer bugs, which may only occur for particular integer values. Molnar described SmartFuzz, a fuzz test generation tool that uses constraint solving to more quickly find inputs that should cause a program to crash.

SmartFuzz performs a symbolic execution of the program under test, yielding a set of constraints on integer variables. These constraints may then be solved to determine the set of inputs that should either be rejected or trigger a bug. In order to generate constraints on signedness, SmartFuzz uses a four-type system in which an integer is either unknown, signed, unsigned, or bottom. If at some point in execution the inferred type of a variable is bottom, SmartFuzz will search for a constraint to assign a negative value to that variable to test for a signed/unsigned conversion bug.

SmartFuzz gives fuzzed inputs to programs running in Valgrind, which will detect any memory errors caused by fuzzing. While effective at determining whether an input causes a bug, this use of Valgrind results in different test cases discovering the same bug and different bugs being discovered by the same test case. This results in multiple reports being filed for the same bug on different test cases. The solution Molnar presented is a fuzzy stack hash which maps the first three frames of a stack trace to a bucket for a single bug. Then a single report is generated for each bucket.

MetaFuzz is run in the Amazon Elastic Compute Cloud (EC2), where CPU time can be rented for 10 cents per hour. The metric used for evaluating fuzz testers in this environment is dollars spent per bug found. The evaluation compares SmartFuzz against zzuf, a block box fuzz tester. The two fuzz testers were run against six programs: mplayer, ffmpeg, convert, gzip, bzip2, and exiv2. SmartFuzz achieved a lower cost per bug than zzuf on two out of six programs and found two bugs in gzip, in which zzuf found none. The metafuzz framework can be accessed at http://metafuzz.com.

Someone asked why the black box fuzzer, zzuf, found more bugs than SmartFuzz on four of the six programs tested. Molnar explained that this is because zzuf changes much more of the fuzzed inputs between tests. This will find more bugs in unrefined code, whereas SmartFuzz is aimed at finding more obscure bugs in mature code. When were most bugs found? Molnar said, early on in the process. How difficult does the analysis become for programs that require complex inputs and user interaction? There was no relationship between complexity of inputs and difficulty, but there are engineering issues that need to be overcome to fully automate the testing of interactive programs.

- ***Nozzle: A Defense Against Heap-spraying Code Injection Attacks***
  Paruj Ratanaworabhan, Cornell University; Benjamin Livshits and Benjamin Zorn, Microsoft Research

Ben Zorn described NOZZLE, a software detection method for heap-spraying attacks. Heap spraying is a method for achieving code execution in the face of address space layout randomization (ASLR). The goal is to place many instances of malicious code on the heap, then jump to some address in the heap with the injected code. Many instances are needed, as ASLR prevents the calculation of the correct address of the malicious code.

NOZZLE provides protection against JavaScript-based heap-spraying attacks in which the malicious code is placed on the heap through the allocation of objects, usually strings. NOZZLE does this by inspecting objects on the heap to determine if they seem malicious (e.g., if they contain a no-op sled, a long series of no-op instructions that lead to executable code). If a percentage of objects are malicious beyond a certain threshold, the offending script is stopped. The simplest way to check if an object is malicious is to check for the presence of a no-op sled, but this technique produces too high a false positive rate. Instead, an object is marked as malicious if it contains a sequence of instructions that looks sufficiently like executable code. This is a hard task in itself, as virtually any byte sequence can be interpreted as x86 instructions.

To detect executable code, NOZZLE uses program flow analysis on objects to determine their attack surface area (SA). The SA of each potential code block in an object is the likelihood that the block is reachable if execution occurs in its containing object. The surface area is then propagated throughout the control flow. Blocks that contain invalid opcodes, such as those that must be executed in kernel mode, have zero SA. NOZZLE exhibits zero false positives and zero false negatives when tested on the 150 Web sites and 12 known heap-spraying attacks, respectively. Note that this is with a 100% sampling rate. In the case of full sampling, NOZZLE causes a maximum overhead of two times normal page-load time, and around 5–10% overhead with a 5% sampling rate. Zorn concluded the talk with a live demonstration in which NOZZLE successfully detected heap spraying.

Avi Rubin pointed out potential means for circumventing NOZZLE, including runtime-initialized objects and code obfuscation with junk data. Zorn pointed out that jumps to code in different objects is another way to trick the surface area calculation, and that they are exploring mitigations for all of the described escalations. Adam Barth (UCB) asked whether NOZZLE would be effective against a less aggressive heap-spraying attack in which only 10% of objects are malicious. Zorn explained that NOZZLE would not detect such an attack, as the malicious surface area is too small, and that NOZZLE is not effective if an attacker is willing to settle for a low success rate.

INVITED TALK

- ***Toward a New Legal Framework for Cybersecurity***
  Deirdre K. Mulligan, School of Information, University of California, Berkeley

Summarized by John Brattin (jbrattin@student.umass.edu)

Deirdre Mulligan spoke about the difficulties involved in designing laws to help protect end users from cyber-attacks. Her main ideas were: a public health analogy may be fitting, and a new legal framework for cybersecurity could benefit from this approach; by using tactics such as mandatory information disclosure, the law could be more flexible than technical standards in regulating software; and because we have a participatory government, people with computer security expertise should get involved in helping design a new legal framework for cybersecurity.

Lack of adoption is a major problem in computer security. There's no point in coming up with new, stronger security practices if no one will bother to use them. "Security in the marketplace is remarkably below what known best practices could provide." In many cases, it isn't even a technical problem—we have the theories, and we even have the theories implemented in software, but people choose not to use the software. Many people use virus protection software but don't update definitions. Many people don't download critical security patches.

Mulligan notes that law is a somewhat unpopular channel for effecting change in the cybersecurity community. People think law moves too slowly, lagging significantly behind changes in technology. Currently, cybersecurity law focuses on deterrence: "increasing the celerity, severity, or certainty of punishment for criminal activity." However, certainty of punishment is remarkably difficult to increase, as cybercriminals are notoriously difficult to identify and, once identified, are frequently not under our jurisdiction. For these reasons, deterrence seems like a poor choice of policy.

Another option is to "incentivize the good guys" to use more secure practices. One way we could get developers to use secure practices is by using notification laws: when a company emits a large volume of toxic waste, they must report it to the government, and eventually the information becomes public. This may lead companies to limit emissions in order to avoid bad publicity. This "mandatory reporting" method prevents the government from directly interfering, but creates incentives for developers to address security concerns.

Cybersecurity will continue to be an important issue; as technology changes and improves, so, too, do technological attacks. We will never completely stop these attacks. We also put ourselves at risk by having an open flow of communication—just as you can defend yourself from biological viruses by staying in your house, you can defend yourself from computer viruses by avoiding the Internet. Another parallel between public health and computer health is that viruses spread in a monoculture. If we had more diversity in systems, particularly operating systems, viruses might not spread as easily.

Public law is a useful channel through which to combat cybercrime, primarily because it is more flexible than using rigid technical standards. By using public law, we can guarantee that a certain problem is addressed by software, or we can guarantee information disclosure that results in greater security, without enforcing the use of any particular system that may quickly become outdated. The law gives another layer of abstraction, in essence. However, people with technical know-how should participate in the construction of appropriate laws.

A member of the audience suggested that people don't patch because patching requires the user to restart, which is time-consuming. She also suggested that it is difficult to enforce security when there aren't clear standards. Mulligan proposed a checklist strategy: if a developer has in some way addressed every problem on the list, she's done her job. Another strategy would be to let developers come up with their own standards and merely report when those standards are violated. Another audience member noted that although diversity may slow the spread of viruses, software becomes more useful the more people use it. He then asked if the government should somehow regulate the development of patches, to make sure nothing breaks. Mulligan expressed doubt that the government would ever interfere so directly in development.

NETWORK SECURITY

Summarized by Ben Ransford (ransford@cs.umass.edu)

- ***Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine***
  Shuang Hao, Nadeem Ahmed Syed, Nick Feamster, and Alexander G. Gray, Georgia Tech; Sven Krasser, McAfee, Inc.

Shuang Hao spoke about SNARE, a system that uses network-level (as opposed to content-level) features of email transmissions to detect spam. Hao cited familiar figures on the cost of worker productivity lost to spam and the prevalence of spam; he also pointed out that spam is increasingly being used as a vector for malware. He stressed the high cost of content filtering in terms of both network traffic and human time. IP blacklists, commonly used in addition to content filtering, are incomplete, and many spam senders are newly infected machines without reputations. The authors' approach in SNARE incorporates empirically derived heuristics to classify email transmissions, given as little as a single packet. Hao asserted that network-level features may be cheaper to analyze than content-level features, because they require less input data; they may also be more difficult for a spammer to vary. Hao offered the basic intuition that, because over 75% of spam is thought to come from botnets, the sending patterns of spammers should be distinguishable from those of human non-spammers.

SNARE uses a set of thirteen features to classify spam. Hao focused on five heuristics that classify messages based on the receipt of a single packet (and, in some cases, using auxiliary knowledge gained from previous interactions with the sender). First, Hao asserted that most legitimate email does not need to travel a long distance geographically from the sender's computer to the recipient's; according to their data set, 90% of legitimate messages travel 2,500 miles or less. Second, because clients participating in botnets tend to share network space with other botnet participants, spam often arrives directly via IP addresses that are numerically close to others that have submitted messages; legitimate messages tend not to exhibit this pattern. Third, legitimate senders and spammers exhibit different sending patterns throughout the day, with spam traffic peaking slightly later than legitimate traffic. Fourth, while legitimate email tends to arrive via mail servers that listen on standard mail-related ports, spam does not; Hao noted that 90% of the spam senders in their data set had none of the standard mail-related ports open. Finally, because some ISPs are more spam-friendly than others—the top 20 ASes in their data set hosted 42% of the spamming IP addresses—the sender's AS may provide a clue to the legitimacy of the message. Hao concluded that SNARE is capable of providing an effective first line of defense against spammers.

George Jones asked whether SNARE consulted lists of known dynamically assigned IP addresses; Hao answered that it did not. Had the authors considered ways to improve on the linear function used to score messages? They considered improvements to the classifier a separate problem. Michael Sirivianos expressed doubt that the geodesic distance feature was equally valid for non-US recipients; Hao agreed that different regions might exhibit different characteristics, and he remarked that the use of several different features made SNARE more robust. An audience member asked which of the classifying features was the best predictor of spam in the authors' experiments. Hao replied that it was the sender's AS number.

- ***Improving Tor using a TCP-over-DTLS Tunnel***
  Joel Reardon, Google Switzerland GmbH; Ian Goldberg, University of Waterloo

Joel Reardon spoke about a way to improve the performance of Tor. Reardon introduced Tor as an overlay system that grants anonymity to anyone on the Internet, most importantly to people who are subject to Internet censorship. The authors propose a change to the way in which Tor routers handle concurrent connections; their change reduces packet delivery latency and, according to the authors, makes Tor more usable.

The authors studied latency in Tor in an attempt to find bottlenecks. Reardon remarked that communication delays—that is, those imposed by throughput limitations—were negligible compared to overall latency. By running a Tor node at a university and exchanging several pieces of data, they eventually found a bottleneck in the buffering strategy Tor uses to multiplex connections. While input buffers drained quickly, output buffers occasionally required packets to wait a long time to be sent. Because Tor uses one socket per router-router link and because the underlying asynchronous communication library, libevent, waits to send on a socket until the operation is guaranteed not to block, data queues up in the output buffers waiting for the socket to become writable. The authors investigated further and found that TCP congestion control was the primary cause of such blocking: if circuits A and B are multiplexed along a link E, then congestion control on E will affect A and B regardless of the respective traffic on each. Reardon showed several graphs illustrating how output buffers on a Tor router changed over time. As an alternative to multiplexing, the authors implemented a scheme in which each circuit that traversed two routers received its own TCP connection between the routers. To avoid several problems (e.g., information leaks) with using TCP directly, the authors tunneled TCP over UDP streams with Datagram Transport Layer Security (DTLS). To prevent clients having to modify their kernels, the authors implemented a user-space TCP stack that can assemble packets suitable for sending via DTLS. Each router advertises a single UDP socket that multiplexes data for all incoming connections; congestion control is performed on a per-circuit basis in the user-space TCP stack. Reardon showed performance graphs demonstrating that Tor with TCP-over-DTLS exhibits much less latency under load than unmodified Tor. Reardon discussed future work involving Tor's new user-space TCP stack and rethinking Tor's buffering strategy.

Michael Sirivianos asked whether it would make sense to make Tor an IP-level service with congestion control only at the entry and exit nodes. Reardon remarked that such a strategy would decrease throughput, because congestion control does not work well on long paths.

- ***Locating Prefix Hijackers using LOCK***
  Tongqing Qiu, Georgia Tech; Lusheng Ji, Dan Pei, and Jia Wang, AT&T Labs—Research; Jun (Jim) Xu, Georgia Tech; Hitesh Ballani, Cornell University

Tongqing Qiu spoke about LOCK, a system that locates IP prefix hijackers by using PlanetLab to monitor traffic to hijacked networks. The Internet comprises tens of thousands of networks (called autonomous systems, or ASes) that exchange packets according to an inter-network routing protocol called the Border Gateway Protocol (BGP). BGP's lack of authentication allows any AS to announce ownership of any other AS, which means any network can hijack, or steal, another network's traffic. Qiu described three kinds of hijacking: blackholing, in which an attacker drops all traffic destined for the victim; imposture, in which the attacker pretends to be the victim such that the victim never receives the hijacked traffic; and interception, in which the attacker transparently interposes her own AS into the chain of networks leading to the victim. Previous approaches to the problem of prefix hijacking have been stymied by the difficulty of changing the Internet's routing infrastructure or have otherwise been focused on recovering the network without pinpointing the source of the error. Qiu claimed that his team's work was the first study of the hijacker location problem. Their system, called LOCK, aims to locate hijackers automatically in order to minimize the effort required for mitigation and recovery.

LOCK uses monitoring software on PlanetLab nodes distributed around the world. Given a network prefix P (owned by a specific AS) to monitor, LOCK's constituent nodes periodically observe the AS paths from their own networks to P. If some monitoring nodes detect that their respective paths to P have changed—Qiu called such nodes "polluted"—they follow an algorithm that infers the location of the hijacker. The algorithm finds the ASes within one hop of the prefix P in a public database of AS relationships. It then considers all the neighbors of ASes on the new paths to P. Because a hijacker cannot manipulate the path to her own AS that traverses her upstream providers, her AS will appear in the set of neighbors, so the algorithm restricts its search to that set. Because paths from polluted monitors (which are distributed diversely around the Internet) to the hijacker naturally converge around the attacker's AS, simply ranking the ASes in the neighbor set by the number of times they appear in paths from polluted monitors allows a quick whittling of the search space. Qiu showed experimental evidence that, for real and synthetic hijacking events, LOCK correctly ranked the hijacker's AS in the top spot up to 94.3% of the time.

George Jones remarked that, although LOCK's detection mechanisms appear sound, the design sidesteps the basic issue that no authoritative central registry of AS relationships is kept up-to-date, thereby making it impossible to determine with total certainty whether a new AS announcement is good or bad. Qiu agreed that the lack of a central registry was a fundamental problem. An audience member asked whether an attacker couldn't simply prepend arbitrary AS numbers into the AS path it announces, and whether that affected LOCK's ability to infer the hijacker's neighborhood. Qiu remarked that some existing routers implement sanity checks that would flag such announcements, but agreed that a hijacker might be able to foil the neighborhood inference by including arbitrary AS numbers in its announcement.

INVITED TALK

- ***Modern Exploitation and Memory Protection Bypasses***
  Alexander Sotirov, Independent Security Researcher

Summarized by Martim Carbone (mcarbone@cc.gatech.edu)

Security researcher Alexander Sotirov, well known for his work on offensive techniques, gave a very instructive talk on the past, present, and future of memory exploitation. His talk also covered the other side of the game by analyzing the evolution of countermeasures, from virtual inexistence to techniques such as Data Execution Prevention and Address Space Layout Randomization (ASLR). Overall, his presentation gave useful information and insight to the audience on the nature of this arms race that has been going on for many years and shows no sign of stopping.

Sotirov started by introducing some basic concepts, such as what exactly constitutes a memory corruption vulnerability and an exploit. The latter is defined as a way to make the target process execute arbitrary code by exploiting the vulnerability. Although these two concepts are commonly coupled, the difficulty of finding a vulnerability and that of effectively exploiting it are not closely related. Sotirov explained that the distance between the two has varied over time, as our understanding of memory exploitation and countermeasures has increased.

In the late '90s, finding a vulnerability could be a hard task, but once found, building a working exploit for it was trivial, given the absence of mitigations. As time passed, techniques for finding new vulnerabilities were systematically improved, reaching a climax in the summer of 2004. At that time, several classes of vulnerabilities were known with no effective mechanisms to counteract them, along with effective techniques for automatically finding such vulnerabilities, such as fuzzing. Examples mentioned by Sotirov include the infamous stack overflow, structured exception handler (SEH), heap overflow, and format string exploitation techniques. As he pointed out, all these exploitation techniques relied on assumptions made about the target process's execution environment. Examples are the fixed locations of code and data regions in memory, a well-known stack layout, and the fact that data placed on a program's stack/heap can be executed as code. The "golden age" of exploitation came to an end as operating systems started being shipped with some basic mitigation mechanisms that invalidated some of these assumptions.

Windows XP Service Pack 2, released in August of 2004, was the first attempt at mitigation and included support for features like disallowing code execution at the stack and heap of programs by leveraging new hardware support (the NX bit), safe heap header unlinking to prevent using unlinking to control execution flow, and stack cookies. Sotirov explained some of the workarounds that were developed to counteract these mitigation techniques and the other mitigation techniques that were developed in response. For example, a simple way to circumvent stack cookies was to no longer rely on overwriting the return address but to use other variables and function arguments. In response to this, compilers started supporting variable reordering in the stack by placing all the buffers at the end of the local stack frame. However, exploitation was still possible by overflowing into other buffers or even the stack frame of the calling function. Exploitation and mitigation techniques involving SEHs were also discussed.

Two state-of-the art mitigation techniques were given special focus in the presentation: Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR). The first effectively closes the window for code injection attacks by disallowing code execution in the program's data regions. The second operates by loading executables at randomized locations in virtual memory, preventing an exploit from correctly guessing the address of its payload. Sotirov explained that DEP and ASLR are only useful when deployed together. Alone, DEP can be circumvented through return-into-libc-style attacks and return-oriented programming, techniques that execute the program's own code in a controlled manner for malicious purposes. If ASLR is used alone, exploitation is still possible by using "heap spraying," which fills the program's heap with copies of the malicious payload, to the point that an exploit writer can say for sure that a certain address will contain a copy of it, despite the randomization. Although included in Windows Vista, these two mitigation techniques had limited impact, since DEP was disabled by default and ASLR was only used for a small set of system services. It is expected that Windows 7's implementation of DEP and ASLR will have much better support for third-party applications.

Due to these new mitigation techniques, the situation at the moment is the opposite of that of the late '90s: finding vulnerabilities has become a relatively easy task, whereas exploiting them now sometimes requires many man-months of hard work, according to Sotirov. In light of this, he moved on to discuss new possibilities in memory exploitation as well as interesting research directions. These include the development of techniques to disclose memory content, which would allow ASLR to be bypassed, as the secrecy of a program's location in memory would be lost. Another one relies on partially overwriting the low-order bytes of pointers, giving an exploit access to the region of the address space occupied by the target process. This works because ASLR randomizes only the 16 high-level bits of addresses, i.e., programs are still 64K-aligned. Entropy attacks against ASLR are also possible. In these, an exploit is executed many times until all possibilities for a program's location in memory are covered. Sotirov also mentioned the possibility of corrupting a program's non-control data as a way to manipulate its internal logic. This attack would require a more detailed understanding of the program's semantics, though. Finally, he proposed as a research direction the use of program analysis techniques to better understand and control a process's memory layout, as a way to oppose ASLR.

Sotirov concluded by arguing in favor of current mitigation techniques, as they significantly raise the bar against exploitation. The question of whether they are enough is hard to answer, but it is likely that the arms race will continue for the time being. And, as he pointed out, "we will always have the Web and all of its brokenness to look at."

Rik Farrow wondered why format string vulnerabilities had disappeared so quickly. Sotirov replied that it was because they were so easy to find. Peter Kristic asked whether using virtual machines (like VMware) makes any difference with regard to exploitation techniques. It makes no difference, since full virtualization replicates the execution environment of a real machine. Ben Zorn asked whether Sotirov had thought about any new mitigation techniques which might help to defend against some of the new attacks, to which he comically replied, "Certainly, but I will not tell you what they are." Sotirov also mentioned (citing the example of Microsoft) that as a result of this arms race, programmers' awareness about writing secure code had increased, but he cautioned the audience never to underestimate the potential of developers to introduce new vulnerabilities into code. And in the unlikely circumstance that all memory corruption vulnerabilities are found and fixed, the Web will always be there as a fertile ground for future exploitation, with whole new classes of vulnerabilities.

JAVASCRIPT SECURITY

Summarized by Ben Ransford (ransford@cs.umass.edu)

- ***Gatekeeper: Mostly Static Enforcement of Security and Reliability Policies for JavaScript Code***
  Salvatore Guarnieri, University of Washington; Benjamin Livshits, Microsoft Research

Ben Livshits explained that Gatekeeper statically analyzes JavaScript code to check for violations of security and reliability policies. Statically analyzing JavaScript is difficult because it offers many ways to accomplish any given task. For example, to materialize an alert box one can call simply call alert(), one can use document.write() to write a call to alert(), one can create an alias of document.write() and call it, one can use eval() to write a call to document.write() that writes a call to alert(), and so on. Gatekeeper allows administrators to set simple policies that disallow certain JavaScript features. It uses a whole-program static analysis approach that is, according to the authors, general enough to be used for purposes other than policy enforcement.

Gatekeeper recognizes two subsets of JavaScript: JavaScript_{GK}, which lacks several JavaScript features including eval(), and JavaScript_{SAFE}, which further lacks several more features. The subsets are such that the SAFE variant is fully statically analyzable without runtime checks, while the GK variant requires basic instrumentation at runtime to aid policy enforcement. Livshits described the authors' experiments on over 8,500 JavaScript widgets from three major Web sites owned by Microsoft and Google, noting that the majority of those widgets were already in the SAFE subset or GK subsets without any need for modifications. Given a program in one of the JavaScript subsets, Gatekeeper uses points-to analysis to track object relationships, thereby ensuring that object aliases do not confound the policy checker. Livshits said that Gatekeeper's points-to analysis is sound, meaning that its policy checker finds all violations it knows to look for. To illustrate the syntax of policy declarations, he showed an example of a Datalog rule that recognizes calls to document.write(). Finally, he offered experimental results: with nine security policies and two reliability policies in hand, Gatekeeper found 1,341 policy violations across 684 of the 8,500 widgets; it also found 113 false positives spread across only two of the widgets.

Adam Barth asked whether Gatekeeper had to parse HTML in order to catch violations; Livshits responded that disallowing document.write() was sufficient to make parsing HTML unnecessary, and that they did not test their system without disallowing it. The session chair, Lucas Ballard, expressed appreciation for the authors' choice of a small, tractable data set (viz., widgets), and asked whether they had attempted to apply their techniques to more complex content. Livshits remarked that analysis of such content was one of the authors' long-term goals, but that most large applications use some of the constructs Gatekeeper flags as suspicious. Livshits suggested that by-hand annotation of legitimate uses of such constructs would be a reasonable way to allow them without confusing Gatekeeper. Ballard proceeded to ask how the authors' work relates to JavaScript strict mode, to which Livshits replied that there were various connections. He remarked that current approaches required subsetting JavaScript and expressed hope that, for the sake of simplicity, some of the current approaches would be implemented directly in the browser.

- ***Cross-Origin JavaScript Capability Leaks: Detection, Exploitation, and Defense***
  Adam Barth, Joel Weinberger, and Dawn Song, University of California, Berkeley

Joel Weinberger spoke about using JavaScript heap-graph analysis to find a previously unnoticed class of browser vulnerabilities. The JavaScript security model includes a notion of contexts, which are separate containers for separate collections of objects. Such separation is designed to prevent private information from leaking between pages or page elements, specifically those with different origins; for example, an advertisement from an ad network should not be able to steal cookies from the page that embeds it. The policy of separating objects from different origins is commonly referred to as the same-origin policy. Weinberger claimed that the authors' work uncovered a new class of vulnerabilities in browsers' enforcement of the same-origin policy.

Weinberger pointed out that browsers implement two concurrent—and different—security models when it comes to JavaScript. Although the Document Object Model (DOM) that exists in each JavaScript context has a reference monitor and a concomitant same-origin policy enforcement mechanism, the JavaScript engine is a separate entity that uses a separate capability-based policy: if you hold a reference to an object, you are granted access to that object. The authors call the circumvention of the DOM's policy in favor of the permissive one a cross-origin JavaScript capability leak: if context B somehow obtains a reference to an object in context A (e.g., if context A passes a reference to context B, or if such a reference leaks), then context B is allowed to access the object without obtaining permission from context A's reference monitor. To detect these capability leaks, the authors instrumented WebKit's JavaScript engine with calls into an analysis library at object creation, destruction, and reference. As the program executes, the library fills out a heap graph.

Weinberger showed several heap graphs of increasing complexity, then described their automated heap graph analysis as a tree traversal which flags edges that span multiple contexts. Running their heap graph analysis on the security-related tests from the WebKit test suite revealed two new vulnerabilities, of one of which Weinberger showed a graphical example. The same technique found several major flaws in the open-source CrossSafe cross-domain JSON request library. Finally, Weinberger suggested access control checks on every object property access as an in-browser defense mechanism, and he remarked that the results of the added checks could be cached using a mechanism that already exists in modern browsers.

Ben Zorn pointed out that JavaScript benchmarks have tight loops that result in access control checks being handled primarily from cache, and he asked whether the authors have tested the overhead of their proposed defense mechanism on code other than test suites. Weinberger responded that the authors have tested other code informally and found their mechanism's performance to be qualitatively good. Lucas Ballard asked Weinberger whether any Web page could exploit the WebKit vulnerabilities to gain access to any other Web page, and Weinberger remarked that before WebKit was patched such exploitation had been possible.

- ***Memory Safety for Low-Level Software/Hardware Interactions***

John Criswell, University of Illinois; Nicolas Geoffray, Université Pierre et Marie Curie, INRIA/Regal; Vikram Adve, University of Illinois

Nicolas Geoffray spoke about SVA-OS, a system that identifies memory safety violations in low-level software-hardware interactions with the goal of defanging kernel bugs. He defined software-hardware interactions as sequences of instructions that manipulate hardware resources. Such interactions, even when expressed in perfectly valid code free of type errors, can circumvent the execution environment's memory safety guarantees or corrupt the hardware resources they manipulate. Geoffray cited processor state, I/O objects, and MMU mappings as examples of manipulable hardware properties whose misuse can result in security violations. As a set of enhancements to the SVA compiler-based virtual machine, SVA-OS comprises a Linux 2.4 instance plus low-overhead compiler analysis and runtime checking of hardware accesses.

Geoffray presented details about how SVA-OS intervenes in several hardware interactions; the interventions are implemented either as special instructions in the SVA VM or as runtime checks. To prevent a task's processor state (e.g., the program counter) from being manipulated before it is properly restored by a context switch, SVA-OS adds an instruction that, instead of temporarily storing a task's processor state in memory where it can be manipulated at rest, atomically swaps one task's processor state for another's. To ensure that memory-mapped I/O operations behave properly, SVA-OS adds I/O-specific load and store instructions whose operation parallels that of regular memory loads and stores; it then segregates memory operations into those that do and those that do not affect I/O. Further, SVA-OS adds runtime checks on MMU updates to ensure that kernel memory is not mapped into user space and that physical memory is not remapped to incorrectly typed virtual pages. Geoffray reported that SVA-OS caught bugs that SVA did not: they tested two real-world MMU exploits, which SVA-OS disallowed; they injected errors into their Linux kernel and observed that SVA-OS prevented crashes; and they discovered that SVA-OS would have caught a serious bug in an Ethernet driver for Linux 2.6 that disabled many network cards. Geoffray showed several performance graphs demonstrating that SVA-OS imposes negligible overhead compared to SVA.

An audience member asked how SVA-OS preserves type safety during MMU remapping and wondered whether SVA-OS maintained type information for physical memory. Geoffray responded that SVA (rather than SVA-OS) did so by segregating physical memory by type.

INVITED TALK

- ***How the Pursuit of Truth Led Me to Selling Viagra***

Vern Paxson, EECS, University of California, Berkeley, and Senior Scientist, International Computer Science Institute

Summarized by Todd Deshane (deshantm@clarkson.edu)

Vern Paxson, a self-proclaimed empiricist, admittedly loves data. The reason he loves data so much is because he has a thirst for the truth and also a phobia about being fooled. In this invited talk, Dr. Paxson described over two decades of Internet measurements and how the changes have been both incredible, at times surprising, and often unpredictable. He started with a general description of network characteristics and then talked about some early manual attacks, followed by the emergence of worms, botnets, and spam. He explained how this led him to begin a campaign to pretend to sell Viagra.

There are three invariants throughout his study of Internet data: growth, explosive onset, and diversity. Between the time when Vern applied to graduate school in 1988 to the publication of his paper "Growth Trends in Wide Area TCP" in 1994, the Internet grew from about 56,000 Internet hosts to about 3 million. The growth was attributed to the explosive commercial use of the Internet, exemplified by WWW traffic doubling every eight weeks from late 1992 to 1994.

Dr. Paxson's first demonstration of explosive onset appears in his quest to understand some seemingly anomalous data that he received regarding USENET bulletin board traffic. His data from 1986 to 1994 shows exponential growth of USENET usage (80% growth per year). Plotting this data on a log linear graph shows a perfect fit to the line. The only problem was that the data ends in 1994, but Vern really wanted to follow up on the data. He conjectured that it couldn't keep growing exponentially; generally, data like that breaks downward (fades gradually before coming to an end), but it turned out that two new data points showed the contrary. After some investigation, he determined that between 1994 and 1996, abuse, in the form of piracy and porn, arrived on USENET and the Internet as a whole. That abuse broke a decade-old invariant (the consistent exponential data growth) upward and not down as would have been expected.

In the mid-1990s, Internet abuse started becoming a major concern. The operators Paxson was in contact with at Lawrence Berkeley National Laboratory (LBL) wanted to know if he could use the data he was collecting to give some insight into the intrusions. Not only did he think that it was possible, but he thought it could be done in real time. This led him to create the Bro Intrusion Detection System (an open source, UNIX-based project that is still actively worked on by Vern and others), which was running 24 hours a day and 7 days a week starting in 1996. Much of the data presented in the rest of the talk was gathered by Bro.

The ability to use Bro at LBL and the ties with LBL operational deployment were "research gold." In particular, from host-scanning data Paxson was able to describe in detail the traffic changes starting with the emergence of the Code Red worm and the beginning of the worm era in 2001. Worms such as Code Red, Nimba, and Blaster were just the beginning, however. Again, using the scanning data, he was able to describe the emergence (around 2002) of what he refers to as auto-rooter tools, more commonly known as bots. At this time there was another significant increase of traffic, which he attributes to malice. Another interesting phenomenon he described was the diversity of the attacks, both in terms of the services attacked and the patterns of when the attacks occurred. For instance, ports scanned included common well-known ports as well as more obscure ports (such as the Sasser backdoor). The patterns of when scanning occurred ranged from heavy traffic during the day, to consistent scanning traffic regardless of time, to scanning the entire Internet at a certain time of day, every day.

In the second part of the talk, he described how he led an effort to infiltrate the Storm botnet and run a spam campaign. The inspiration for the spam campaign was the fact that he had studied the enemy and understood that profit was the motive of the botnet masters. The shift from curiosity and fame to an underground botnet-based economy had begun. He showed screenshots of professional spam software, sites that auction stolen eBay accounts, sites that sell social networking bots (with separate services that would integrate CAPTCHA bypassers), and affiliate programs that allow people to refer others and get a cut of the profits on these malicious tools. He realized that a large part of the business model was based on turning exploits into bots, then turning the bots into spam worker threads, then converting user clicks into sales. The spam campaign is described in further detail in "Spamcraft: An Inside Look at Spam Campaign Orchestration" presented at LEET '09. He continued by highlighting the fact that spam-filtering software and blacklisting spam bots filtered much of the spam to junk mail folders, which meant that only a small percentage of the spam was actually seen by users in their inboxes. During what he calls their spam conversion experiment (counting fake sales of Viagra) they were able to instrument 1.5% of the Storm botnet workers. They estimate that if they had been able to instrument the entire botnet army, they would have been able to make around $3 million. He notes that there was a lot of FUD (Fear, Uncertainty, and Doubt) about the Storm botnet in the news, where the media made claims of very large profits from Storm (orders of magnitude larger than reality) that are erroneous due to flaws in measurement methodology.

Paxson concluded with some reflections on the enormous changes he has seen in the Internet in just a couple of decades, especially in cybercrime (for profit) and the latent threat of cyberwarfare. He emphasized that measuring is easy, but measuring in a meaningful and sound way is hard (full of unfun grunt work dealing with messiness and error). Despite the challenges, he argues, it is the only way to get the truth and you can even run into some very interesting surprises (including diversity, exponential growth, unexpected threats, and rapid changes in the landscape). He encouraged the students in the audience to take on the challenge, as there is a deep fundamental need for well-grounded empirical data in the computer security field.

An audience member asked whether he thought that more success might come with more waves (repeat customers) of the spam campaign, to which he agreed that it was possibility, but he also noted that there is a tension over whether the botnet would be able to go after these follow-up sales or if the pharmaceuticals themselves would follow up. Steve Bellovin wondered about the ethics and IRB process involved. Paxson responded that he had lawyers look at the experiment, but that it didn't go through the IRB process, although he admits that it should have. A second follow-up study is currently going through a long IRB process, he noted. He also mentioned that there is an upcoming workshop that focuses on ethics at Financial Cryptography and Data Security '10. An admirer of the Spamalytics paper asserted that spam makes a lot of money, to which Paxson responded that he would have thought it would have been more (not only around $2 million per year as according to his data). He recommended that people think about the problem of network saturation (a "tragedy of the commons" scenario). Were the phishing attacks from the same players? He didn't know and speculated about the structure of the attackers, whether there were one or a few kingpins, or if there were, instead, a lot of ankle biters.

RADIO

Summarized by Italo Dacosta (idacosta@gatech.edu)

- ***Physical-layer Identification of RFID Devices***
  Boris Danev, ETH Zürich, Switzerland; Thomas S. Heydt-Benjamin, IBM Zürich Research Laboratory, Switzerland; Srdjan Căpkun, ETH Zürich, Switzerland

RFID chips are important components in the security of systems such as electronic passports (ePassports) and identity cards. Three security mechanisms have been defined to protect ePassport RFID chips, but only one is required by the standards. On the other hand, multiple attacks against these mechanisms have been published by security researchers. These attacks against ePassports prompted the authors to determine whether RFID chips can be uniquely identified based on their physical-layer features and the accuracy of the identification techniques. As Boris Danev noted, this work attempts to achieve a form of hardware biometrics. A direct application of this technique will be the prevention of cloning attacks against ePassports.

Danev described the experimental setup and the different experiments used to collect features from the RFID chips. A total of 10 ePassports and 50 Java cards were analyzed. The authors used three techniques to analyze the data collected: time, modulation shape, and spectral features analysis.

From these techniques, the analysis of modulation shape and spectral features were found to be the most effective (spectral features in particular). Based on the analysis, the authors were able to identify the ePassports' country and year of issuance, and some model and manufacturer chips design. The techniques evaluated also showed good accuracy: a 95% successful identification rate (5% equal error rate). In addition, Danev said that the accuracy can be improved dramatically through the combination of burst and sweep techniques. Finally, the authors have done some preliminary work to determine how hard it is to reproduce the fingerprints to defeat the identification techniques proposed.

Danev was asked if environmental conditions such as temperature and age could affect the fingerprinting of RFID chips. Current work is trying to determine if aging can affect the proposed identification techniques, and more work is needed to analyze the impact of other environmental factors. Several members of the audience were concerned about the privacy risks of fingerprinting RFID chips, such as the remote profiling of individuals. Danev mentioned that such privacy attacks may be possible but not by using the features analyzed in this work, because such features cannot be measured reliably from a distance. Another question related to the use of physical protection mechanisms (i.e., metal shields) in current ePassports. The author mentioned that some techniques are being implemented, but they vary from country to country. Finally, in response to a question regarding the relationship between the quality and the variability of an CRFID tag, Danev mentioned that the cheaper the design the more variability an RFID chip will have and, therefore, the easier it will be to identify.

- ***CCCP: Secure Remote Storage for Computational RFIDs***
  Mastooreh Salajegheh, Shane Clark, Benjamin Ransford, and Kevin Fu, University of Massachusetts Amherst; Ari Juels, RSA Laboratories, The Security Division of EMC

Computational RFID (CRFID) tags introduce a lot of interesting possibilities due to their additional components: a micro-controller, flash memory, and one or more sensors. However, these devices are also affected by hardware constraints: small memory size, tiny energy reservoir, and reboots every few seconds. Using the fact that radio transmissions are cheaper than writing to flash memory, the authors proposed outsourcing storage to a reader to save energy. However, using remote storage presents several security challenges: the data is transmitted over the air and the reader may not be trusted. Shane Clark introduced a new protocol, Cryptographic Computational Continuation Passing (CCCP), that adds the minimum security guarantees to allow CRFID tags to use a reader as a remote storage mechanism in an energy-efficient and secure way. The main goals of the protocols are to use remote storage to get real computational progress out of these devices and to eliminate Sisyphean tasks that result from the short power cycles of the CRFID tags.

The authors outlined a set of security goals: confidentiality, integrity, authentication, and data freshness. Based on these security goals and the CRFID tag constraints, the authors defined some basic and efficient security primitives: the use of stream ciphers (XOR operations) for confidentiality, and universal-hash-function-based MAC (UMAC) for integrity and authentication. To support the use of stream-keys, the authors introduced pre-computation when the tag is idle (good power season) to create stream-key bits. For data freshness, the authors used a unary encoding technique (hole punching) which allowed a counter in memory to be updated more efficiently.

Clark described the experimental testbed used to evaluate CCCP and the methodology followed. Energy was chosen as the most appropriate metric during the evaluation. The main result of the evaluation was that using radio for secure remote storage is cheaper than using local storage up to a data threshold size of 96 bytes. Finally, Clark suggested some future work in this area: the development of more efficient CRFID tags, extensions for long-term storage, work on WOM codes, and a public key system for CRFID tags.

Clark was asked about his expectations regarding flash memory costs in the future. Clark commented that backscatter transmissions used only one transistor, while flash memory operations used several, and that he was quite confident that in the near future flash will not be cheaper than radio. What about atomicity issues with data transmission in the CCCP protocol? A solution to this issue was to increase the counter in two steps: one before and one after data is sent.

- ***Jamming-resistant Broadcast Communication without Shared Keys***
  Christina Pöpper, Mario Strasser, and Srdjan Čapkun, ETH Zurich, Switzerland

It is a well-known fact that RF communications are vulnerable to jamming attacks. Traditional defenses against this type of attack are the use of spread spectrum (SS) techniques such as frequency-hopping SS or direct sequence SS (DSSS). These techniques rely on the use of a shared code to spread the transmitted messages. However, these techniques do not work well on broadcast communication scenarios where a sender wants to broadcast one or more authenticated messages to a potentially large number of receivers and some of the receivers may be unknown or untrusted (e.g., emergency and navigation systems). This paper presents a novel technique, Uncoordinated-DSSS (UDSSS), to solve this problem. UDSSS uses DSSS communication but releases the requirement of a shared secret key by using randomization and the following key observation: "Whatever has arrived unjammed at the receiver can be decoded." To an attacker, UDSSS looks similar to DSSS. The difference is that the code sequence used to spread the messages is chosen randomly from a set of public code sequences that both the sender and the receiver know. The receiver records the spread messages and tries to de-spread them using the public code sequences in a trial-and-error fashion. For successful de-spreading of the messages, it is important to choose the same public code sequence used by the sender, as well as the right synchronization.

Pöpper described the prototype implementation of UDSSS, based on Universal Software Radio Peripherals (USRP) and GnuRadio, as well as the experimental setup and methodology used. Several adversaries were considered during the analysis, using the jamming probability with respect to a given message transmission. Also, message transmission time was used as the main metric during the evaluation. The results show that increasing the processing gain is much more harmful for the message throughput than increasing the size of the public code set, and that message throughput increases with the use of large message sizes. While UDSSS has lower performance than DSSS, it can be enhanced to achieve similar performance to DSSS in the absence of jamming; through the use of two parallel transmissions, one using a single code sequence and the other using normal UDSSS. Pöpper also suggested an optimization that is not described in the paper: using UDSSS to transmit only the spreading code and not the message, which allows faster decoding times and larger message sizes. Finally, Pöpper described a practical application of UDSSS in a navigation broadcast system.

INVITED TALK

Summarized by Salvatore Guarnieri (sammyg@cs.washington.edu)

- ***Designing Trustworthy User Agents for a Hostile Web***
  Eric Lawrence, Senior Program Manager, Internet Explorer Security Team, Microsoft

Eric Lawrence learned a lot while working on Internet Explorer 8 (IE8), and he talked about how IE was designed, where the current threats are, and the future of Web security.

Internet attacks are always evolving, so mechanisms to prevent or limit the effectiveness of these attacks must evolve as well. In Internet Explorer 7 (IE7), the goal was to reduce the attack surface on the local machine. This meant that IE7 had fewer vulnerable areas than previous versions of the browser. Now the local machine isn't as valuable a target. Much data lives in the cloud, so cross-site scripting (XSS), cross-site request forgery (CSRF), and other similar attacks are major problems. IE8 tries to address these new types of attack that don't necessarily target the local machine but, rather, the way in which confidential or high-integrity data is handled in the browser.

Security is difficult for the Web because the space is very complex. The browser needs to be secure and Web developers need to produce secure Web sites. This is a problem, since some Web developers don't even understand what same-origin policy is. Furthermore, security for the Web was largely an afterthought, and many of the interesting security models for the Web don't fit how the Web is being used today. Finally, since Internet Explorer (IE) has been around for a while, users expect that things that worked in an old version will work in a new version. This means that changes to the browser cannot break backward compatibility. It turns out that if backward compatibility is broken, developers don't update their sites to work in the new browser, so users simply refrain from upgrading and are left with a less secure browser visiting possibly insecure pages.

The security team for IE is focused on three areas: (1) security feature improvements, (2) secure features, and (3) security and compatibility. Security feature improvements are new features (e.g., the XSS filter) that exist solely to improve security. "Secure features" refers to the process of ensuring that new features (e.g., IE8's accelerators) are secure and do not increase attack surface. Security and compatibility focuses on ensuring that the security features are compatible with Web sites. Users need the security feature to work on the Web sites they visit or they will roll back to a less secure browser that does work.

IE8 has some features that make attacks much less effective. Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR) are turned on by default. This is a huge help, because they greatly limit the exploitability of memory-related vulnerabilities in the browser. Some new features improve security by reducing the amount of native code needed in the system. Plug-ins such as browser toolbars provide some functionality to users but are usually written in conventional languages like C and often contain security bugs. IE8 has accelerators and Web slices, which allow third parties to enhance the user experience without introducing potentially buggy native code.

No matter how secure the browser is, there is still a user involved in the system. Users are faced with social engineering attacks, such as phishing and malware-distribution sites, that trick them into granting the attacker permission to perform unwanted actions. People have tried a lot of things to warn users about potentially unsafe actions. IE8's SmartScreen Filter feature can provide a bold, unambiguous warning to block known-unsafe actions because Microsoft actively searches out dangerous content on the Internet and flags it using the URL Reputation Service.

SECURING WEB APPS

Summarized by Shane Clark (ssclark@cs.umass.edu)

- ***xBook: Redesigning Privacy Control in Social Networking Platforms***
  Kapil Singh, Georgia Institute of Technology; Sumeer Bhola, Google; Wenke Lee, Georgia Institute of Technology

Kapil Singh gave a talk about the problems with privacy control in social networking platforms and xBook, a system designed to allow fine-grained and reliable control over applications for such platforms. Singh noted that social networking sites such as Facebook, Twitter, and Orkut are still growing rapidly in popularity. Many sites are also evolving into platforms that expose APIs to third-party developers, giving any developer's application access to almost all stored user data. A user must generally agree to this access when adding an application, but there is no guarantee that the application will use exposed data appropriately or that it will only access data that is necessary for its advertised functionality. The goal of xBook is to provide privacy protection to users without requiring changes to the browser or any discernible usability changes from the user's standpoint.

Singh next introduced the high-level approach taken with xBook. All applications are run from within the trusted xBook domain to allow mediation. Applications are then monitored at runtime in the browser. While they are still allowed access to any user data, applications are required to make the use of such data explicit to the user; xBook is able to enforce this policy by tracking information flow. xBook also requires applications to be split into client and server components that are confined appropriately. For example, JavaScript must be written in a safe subset of the language called ADsafe and not have access to a page's DOM elements. To enable necessary DOM accesses, xBook implements a DOM wrapper that gives client-side components access to only those elements that belong to the same application. Even when the components of a single application are communicating among themselves, the data that they are able to exchange is subject to the explicit restrictions specified by the application developer and agreed to by the user. Application components are also subject to these restrictions when communicating with external entities. The authors implemented xBook as a Facebook application to demonstrate its effectiveness, and they also implemented two applications that ran on top of xBook.

An audience member asked if the authors have performed any experiments to test usability. Singh answered that they have not completed any experiments but have designed the system to be easy to use. Another audience member wanted to know why users should be more willing to trust xBook with their data than any given application. He was also concerned that application developers may not be willing to target xBook because they want direct access to user data without added restrictions. Singh responded that users already trust a large number of other applications, all with the same privileges. Trusting xBook only requires users to add a single application, minimizing the number that must be trusted overall. If users choose to use xBook, developers will be forced to target xBook with their development in order to achieve widespread use. Finally, an audience member pointed out that developing applications for xBook is not necessarily an indicator of ease of use. He asked whether the authors had attempted to port an existing application and, if so, what their experience was. Singh countered that the two applications they implemented mirror the functionality of existing apps, but he acknowledged that there may be a learning curve for their cross-platform APIs.

- ***Nemesis: Preventing Authentication & Access Control Vulnerabilities in Web Applications***
Michael Dalton and Christos Kozyrakis, Stanford University; Nickolai Zeldovich, CSAIL, MIT

Michael Dalton presented his work on Nemesis, a system designed to automatically prevent authentication and access control vulnerabilities in Web applications. Dalton first introduced some of the failings of Web authentication with an illustrative example. If a user Bob uploads a photo to a Web application, for example, that photo is typically stored in the database using the credentials of the Web server user. The database has no knowledge of Bob, and the Web application's user account necessarily has privileges equal to or greater than those of any user that exists in the application. According to Dalton, this semantic gap fundamentally breaks Web authentication by requiring the application programmer to accurately insert access control and authentication checks before every file-system or database operation. This must be done perfectly in order to adequately secure any Web application.

Dalton next introduced the two classes of attacks that Nemesis addresses and the approach to preventing each of them. Authorization bypass vulnerabilities occur when a user is able to access a resource without authorization, often as the result of a missing or incorrect authorization check. Authentication bypass vulnerabilities occur when successful authentication occurs without valid credentials, often caused by a poor URL/cookie validation method or simply weak cryptography. Nemesis stops both classes of vulnerabilities by inferring when authentication is performed correctly using dynamic information flow tracking (DIFT), also known as taint tracking, and automatically enforcing access control lists (ACLs) for Web application user accounts (as opposed to Web server processes). To support the use of Nemesis, DIFT functionality must be added to the language interpreter in use, and ACL enforcement must be added to the language's core library. Nemesis does not require the modification of the application code.

A Nemesis prototype has been implemented for the PHP language and Dalton presented some examples of vulnerabilities in real-world applications that Nemesis is able to prevent. The prototype stopped authentication and authorization bypass vulnerabilities in six popular PHP applications without any discernible performance overhead, but does suffer from some limitations. There is currently no SQL query rewriting support, forcing the authors to manually insert code to perform ACL and authentication checks in some cases. The current prototype also lacks automatic ACL generation, which leaves application administrators to write their own. Both SQL query rewriting and automatic ACL generation based on logs are slated for future work.

Bryan Parmo asked if Nemesis could be easily extended to work with sites that use authentication methods other than passwords. Dalton answered that Nemesis currently relies on a byte-by-byte comparison of two strings in order to infer successful authentication, but as long as there is some authentication mechanism that Nemesis can be made aware of, it should be possible to support that method. Parmo followed up by asking if Nemesis is effectively removing the responsibility for authentication from the Web application and handling it completely. This is possible but Nemesis actually requires some amount of ground truth in the application that it can trust as a valid authentication technique. Finally, David Wagner asked if Dalton believed that Nemesis is applicable to other popular Web application languages and if he foresees any particular challenges in supporting these other languages. Dalton answered that many popular languages have had DIFT support implemented for them in the past and it should be straightforward to add Nemesis support. The authors chose to use PHP because they found the interpreter easy to modify, and there are many insecure PHP applications currently in use.

- ***Static Enforcement of Web Application Integrity Through Strong Typing***
  William Robertson and Giovanni Vigna, University of California, Santa Barbara

William Robertson noted that Web application vulnerabilities make up more than half of all reported vulnerabilities over the past two years, according to Symantec. The majority of these Web vulnerabilities are either cross-site scripting (XSS) or SQL injection vulnerabilities. He acknowledged that there are a number of existing solutions such as application firewalls, automated code analysis, and penetration testing, but argued that these are clearly insufficient measures, considering the continued prevalence of XSS and SQL injection vulnerabilities.

Robertson said that a major source of Web application vulnerabilities is the treatment of Web documents and database queries as unstructured character sequences. Because there is no knowledge of appropriate structure and content at the language or framework level, developers are responsible for manually sanitizing this content. Developers will inevitably fail at this task, as correctness requires perfect code. In order to address this problem, the authors implemented a language-based solution intended to explicitly denote structure and content using a strong type system. This approach shifts responsibility for integrity enforcement from the developer to the language and removes the need for separate testing or analysis to ensure that an application is secure.

The authors' prototype solution took the form of a Haskell-based application framework, and they presented evaluation results. The framework enforces integrity by requiring the application to construct a tree of document nodes that it then "renders" into serialized raw text for display. Each node of the tree is sanitized during the rendering process in order to prevent XSS vulnerabilities. The framework is able to remove SQL injection vulnerabilities by simply requiring applications to exclusively use prepared statements where possible. When prepared statements are not possible, the framework enforces integrity using a node rendering approach similar to that for document nodes. Robertson claimed that their framework achieved full sanitization coverage and that they had confirmed correct sanitizer operation using a large number of randomly generated test cases. Finally, Robertson showed that their framework's performance was similar to that of Tomcat and Pylons. They found that the performance of their prototype fell between the other two frameworks in testing and is thus acceptable.

Adam Barth asked if the authors have investigated the possibility of removing XSS vulnerabilities stemming from client-side code such as JavaScript. Robertson answered that they are investigating this problem. One promising approach might be to compile a representation of the client-side code on the server side into JavaScript, similar to Google Web Toolkit's approach. Benjamin Livshits asked about using symbolic execution to verify sanitizers, as opposed to random testing. The authors have not looked into this, but it seems like a useful approach. Finally, David Wagner complimented Robertson on the system's removal of the ability to even express some server-side vulnerabilities and asked if the authors have looked into any other classes of vulnerabilities. They chose to focus on SQL injection and XSS because they can be simply expressed and prevented. The authors are considering other vulnerabilities, but they may be more difficult to address.

INVITED TALK

- ***Compression, Correction, Confidentiality, and Comprehension: A Modern Look at Commercial Telegraph Codes***
  Steven M. Bellovin, Professor of Computer Science, Columbia University

Summarized by Kevin Butler (butler@cse.psu.edu)

Steve Bellovin delved into historical archives to present an enlightening investigation of telegraph codes, looking back with a modern perspective and uncovering remarkable similarities to our current network structures and protocols. An example of these surprising findings was discovering that telegraph systems contained protocol stacks—the link layer was well defined—and error correcting codes. Some mechanisms were different, e.g., the job of a router was performed by an operator who would relay telegraphs to the correct destination link, but many of the problems were similar to those we currently face. Steve focused on four areas: compression for reducing the cost of transmission; detection and correction of coding errors; confidentiality to deal with operators seeing messages as they are transmitted; and comprehension of the culture of the time, which is strikingly conveyed in how the codes were structured.

The talk described a succession of codes from Sir Home Popham's naval code in 1805, which was expressive enough to allow conversation, through to the telegraph era, where codes were compiled not only for general use but for specific industries as well. As an example, theater codes differentiated between specific capabilities desired of chorus girls, each description compressed to a single code word. It was noted that we still use domain-specific compression: Lempel-Ziv doesn't work as well on multimedia as JPEG or MP3. Early codes did not provide redundancy, and techniques such as terminal indices and mutilation tables were created. These would provide potential endings for words where the first two letters were known, and usually about five possibilities were available. Based on the sentence, the code could either be retransmitted or disambiguated contextually.

Confidentiality was an issue, identified in an 1870 document that described a threat model for communication security. However, it was not particularly well implemented in many codes. Techniques included mono-alphabetic substitution and constant additives. Even the US Government's code consisted of constant additives built on top of the Western Union code, and this was state of the art until around World War I. Bloomer's Commercial Cryptograph was a more successful code, a "holocryptic cipher" that was unreadable without a key. This was a user-created two-part code.

Some interesting details also appeared about threats to confidentiality, such as Britain requiring cable companies to turn over copies of all international telegrams, and copyright, where the US was not a signatory to the Berne Convention until 1976, causing publishers to often publish their works first in the US to ensure copyright there before publishing in the rest of the world. A particularly interesting reference to copyright infringement as "piracy" appeared in a code book from 1936.

Telegraph codes were only recently supplanted in many places, with China using them regularly until around the year 2000. Much of what can be found in modern codes is an evolution of what was done then, with formalized and mathematical models now created, but the basis was set for these techniques at least 75 years previously. A draft paper of these findings can be found on Steve's Web page at http://www.cs.columbia.edu/~smb/papers/codebooks.pdf.

In the Q&A session, Matt Blaze pointed out that some early military codes were based on poetry, and that codes are only as secure as the people compiling them. Steve added to this by discussing how early Russian OTP codes were not in fact random; their randomness depended on the people generating the pads.

APPLIED CRYPTO

Summarized by Andres Molina (amolinaf@nsm.umass.edu)

- ***Vanish: Increasing Data Privacy with Self-Destructing Data***
  Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy, University of Washington

  Awarded Outstanding Paper!

Roxana Geambasu made the case that currently, when data is communicated over the Internet, it lives forever. Moreover, while it is in transit, different providers create multiple copies of the data. The users do not know where these copies reside or for how long they will be kept. Some of this data may be sensitive and resurface in a manner that is unwanted by the user; for example, by means of a subpoena. The speaker referred to this type of attack as a retroactive attack on archived data. She also pointed out that current solutions such as PGP do not protect against such attacks, because the key to recover the archived material can be obtained by legal means. This problem also occurs in centralized services in which a provider maintains the data in an encrypted form. The speaker gave the example of Hushmail, an encrypted email provider that, in at least one case, allowed access to email content as required by a subpoena.

Vanish is a system that combines distributed hash tables (DHTs) together with secret sharing techniques to accomplish the goal of destroying data. The data gets encrypted using a key that is split using Shamir shared secrets, gets stored using the DHT in key-value pairs, and gets destroyed after the data is successfully encrypted. Properties of DHT systems such as member churn and discarding of key-value pairs after a timeout is reached allow the proper destruction of the shared key after an expiration date without requiring any action by the user. With Shamir shared secrets, a threshold amount of the shared key must be recovered, and once that threshold can no longer be obtained the key is gone forever. An attacker would need to capture the key before or while it is being distributed, but this only applies to premeditated attacks, not retroactive attacks.

The system has currently been realized using a Firefox plug-in as the front end, allowing users to interact with services such as Gmail, Hotmail, Facebook, Google Docs, or essentially any other Web application that deals with plain text. However, the speaker mentioned that Vanish can easily be implemented as a plug-in to Thunderbird or other email clients. The presentation only addressed some aspects of the evaluation relevant to security, referring the audience to other details of evaluation criteria in the paper.

A member of the audience asked about the selection of the parameters from the user's perspective. Geambasu responded that the current implementation provides the user with reasonable default parameters that can be modified to balance security and performance. Someone noted the reference in the talk to the need to use encryption techniques like those employed in PGP to protect the data in transit. Geambasu said such encryption techniques should be used in addition to Vanish, not instead of it. Someone pointed out that there are other existing solutions to the problem, such as the Tahoe distributed file system, although this system could pose a major drawback in terms of usability. Geambasu was not aware of Tahoe and agreed that usability was a major consideration in the design of Vanish. How does Vanish protect the anonymity of the recipient, since if the recipient is known, it would be possible to subpoena the recipient's computer in order to obtain the contents of the caches? The Vanish model assumes that the attacker would not know about the sender or recipient until after the data has expired and enough time has elapsed for the data to be destroyed.

For more information and to obtain the application, see http://vanish.cs.washington.edu/.

- ***Efficient Data Structures for Tamper-Evident Logging***
  Scott A. Crosby and Dan S. Wallach, Rice University

Scott Crosby started his talk by noting that everyone has logs and that there are many applications that require storage using tamper-evident techniques. Some of the examples that were given included HIPAA regulations and credit card payment contracts. Scott also pointed out that current commercial solutions to this problem rely on the correct operation of the appliances and are too slow.

Crosby then proposed the use of efficient data structures for tamper-evident logging. This approach would allow a third party to prove the correctness of the solution using known cryptographic techniques. Additionally, this solution offers logarithmic performance in all operations, instead of the linear performance that previous solutions offer. The proposed solution based on history trees would offer performance of 1,750 events per second, including digital signatures, and 8,000 audits per second.

Crosby explained that audits are essential for this type of application. If a malicious logger could anticipate that a particular log entry was not going to be audited, then he could easily replace that entry. For that reason, it is necessary to ensure that every event has a non-zero probability of being audited. Furthermore, the author distinguished two types of audits: audits to verify correct insertion of events and audits to verify consistency among commitments. This new paradigm requires that both insertions and audits are cheap in terms of CPU, communication, complexity, and storage. The solution proposed makes this possible by using Merkle binary trees that avoid having to compute linear chains of hashes. The solution allows the probabilistic detection of tampering by only verifying a subset of events. Additionally, this solution allows the computation of Merkle aggregates and the performing of safe deletion by using tree pruning techniques.

The system was evaluated with a syslog implementation, logging 4 million events using 11 hosts over four days. The experiment was repeated 20 times, using DSA signatures and SHA1 hashes. While the results were reasonable (1,750 events/sec; 8,000 audits/sec), approximately 83% of the runtime was spent computing signatures. Also, compression reduced performance by about 50%. The authors suggested that concurrency and replication would improve performance. Additionally, given that the major bottleneck is due to the computation of signatures, a performance gain is expected if only a subset of the commits are signed, a faster public key encryption scheme is used (such as ECC), or the computation of the signatures is offloaded to other servers. Using the latter approach, it is possible to process up to 10,000 events/sec.

Someone asked if it would be possible to perform data reduction preserving the tamper-proof properties. Yes, this is possible by employing tree pruning techniques and appropriately using Merkle aggregation predicates with each event. Another member asked if the technique would account for trees with different depths, to which the author responded affirmatively.

- ***VPriv: Protecting Privacy in Location-Based Vehicular Services***
  Raluca Ada Popa and Hari Balakrishnan, Massachusetts Institute of Technology; Andrew J. Blumberg, Stanford University

Raluca Popa presented VPriv, a system that can be used in various transportation systems to compute functions over driver paths, such as usage costs. These computations are performed while preserving users' privacy. Current systems such as EZPass do not offer adequate levels of privacy, as it is possible to track the times and places where an EZPass has been used. The solution came from observing that it is possible to compute costs associated with a path without actually knowing all the details of the path, by using zero-knowledge proofs and secure multi-party computation. VPriv was designed from the start to be efficient by exploiting the properties of homomorphic encryption and families of random functions. The system relies on the ability to compute functions on tuples containing times, locations, and random tags unique to each tuple.

The security model considers two parties: a client, such as a driver, and a server. The client is not trusted, as he or she has incentives to alter the protocol in order to reduce personal costs. The server, on the other hand, is partially trusted. That is, the server will be trusted to perform the protocol correctly, but it will not be trusted to preserve the privacy of the client. The speaker mentioned that the paper also discusses a protocol dealing with a malicious server; in other words, a server that is not trusted even to perform the protocol correctly. While the talk focuses on the application of usage tolls, the speaker briefly explained how the system can easily be customized for other applications such as "pay-as-you-go" insurance and speeding violations. The protocol was divided into three phases. In the first phase, referred to as registration, random tags are created to which the client commits. The second phase involves the uploading of the tuples containing the random tags generated in the previous phase, together with the locations and times. Finally, the reconciliation phase is when the client would compute the owed cost and allow the server to verify the result without compromising the privacy of the client.

The evaluation of the protocol showed linear performance on the number of tags and the number of tuples uploaded to the server. The system is three orders of magnitude faster than Fairplay, a state-of-the-art general-purpose secure multi-party computation compiler. Raluca mentioned that it is possible to serve roughly a million customers with 21 server cores. The system was evaluated on Cartel, an MIT project, with 27 taxis and 4,826 one-day paths. The experiments showed that the enforcement scheme is effective and efficient. The authors hope to see this system implemented in the car sharing company Zipcar.

In a system like this, will it be hard for a customer to dispute a bill when there is no actual record of the trips on the server side? This is handled by the reconciliation phase, in which the fees due are actually computed by the customer and only verified by the server. Is it possible to leak some information by allowing the server to identify where the tuples are uploaded from? The system may be complemented with the use of an anonymizing system, such as TOR, to upload the tuples. Ian Goldberg asked if more complex functions could be used to compute costs beyond simple additions. It is possible to compute conditional functions, and, in theory, any polynomial function could also be computed using similar techniques, but the details would have to be explored in future work. In the security model, although the server is trusted to perform the protocol correctly, it may also have incentives to prevent the protocol from being performed correctly. The paper includes an extension to the system that takes into account a malicious server. In such a case, efficiency would be affected.

INVITED TALK

- ***Top Ten Web Hacking Techniques of 2008: "What's possible, not probable"***
  Jeremiah Grossman, Founder and CTO, WhiteHat Security

  Summarized by Stephen McLaughlin (smclaugh@cse.psu.edu)

Jeremiah Grossman presented the top 10 Web hacking techniques of 2008. Tenth is Flash parameter injection (FPI). FPI uses embedded Flash media as a vector for accessing the HTML content of the containing page. One method for FPI requires that an attacker assign values to global variables which are initialized from values in HTTP requests.

Another technique that leverages a browser plug-in, ActiveX repurposing, comes in at number nine. In this attack, a malicious Web page takes advantage of the update functionality in the Juniper SSL-VPN client to first save a malicious configuration file to a known location, then place a path to it in the ActiveX control's INI file.

The eighth Web hack, tunneling TCP over HTTP over SQL injection, uses the reDuh server and client to give an attacker control of a remote machine behind a firewall. In this attack the reDuh server, which tunnels TCP traffic over valid HTTP commands, is uploaded to an SQL server behind a firewall that only allows HTTP traffic. The SQL server is used as a gateway between the attacker and the internal network. While Grossman pointed out that database hardening is one defense against this attack, the reDuh server may also be uploaded to an application server as a JSP page.

Web hack number seven can be used to determine if a victim is logged in to a given Web site or by observing the stylesheets from another Web page. A link to a target inline stylesheet from a malicious page can be used to obtain the cross-domain style definition. The malicious page can then check for style properties indicating the user is logged in to the target site. The sixth Web hack, abusing HTML 5 client-side storage, uses any cross-site scripting (XSS) vulnerability to either leak information from or inject code into HTML 5 client-side stored objects. Web hack five is a different Opera. The goal of this attack is to execute code in the Opera browser running in the opera:* context, which will give an adversary the ability to modify browser settings with opera:config. This is achieved by a cross-site request forgery to opera:historysearch, which already contains an XSS from a previously visited page.

Clickjacking, the fourth Web hack, uses the CSS opacity property and JavaScript to place an invisible button pulled from a form on a target page directly under the user's mouse. This causes the user to click the invisible button instead of some visible part of the page. If the user has already authenticated to the target site, this will result in the site taking some action on the attacker's behalf. Examples of target sites may be CPC advertisements, Digg links, and options on DSL router configuration pages. Grossman went on to say that this exploit may be used to trick users into enabling a laptop's camera and speakers through a Flash applet, allowing for remote surveillance through a Flash application. Grossman had a recommended countermeasure to the camera-enabling attack: placing tape or a sticker over the camera lens.

The third Web hack uses a feature of Safari in which any file of a type not recognized by the browser is saved to the desktop. This allows malicious site to effectively "carpet bomb" the target machine with garbage files or malware.

Coming in at second place is an attack on Google Gears which bypasses the cross-origin security policy. This is possible if an attacker can insert Google Gears commands into a file uploaded to a target site. These commands are likely to pass input filtering, as they lack suspicious tokens like <script> tags. A malicious Gears-enabled site then executes the commands in the context of the target site in the victim's browser.

The number one Web hack of 2008 is the GIFAR, a concatenation of a GIF image and a Java JAR archive. Because GIF files are parsed from the first byte and any garbage at the end is ignored, and JAR files are the exact opposite, appending a JAR file to the end of a GIF creates a GIFAR, which will pass input validation for file uploads while allowing the JAR file to be embedded in a page that will run in the targeted site's context.

Several of the questions after the talk concerned classifying Web hacks by type, technique, and trends. Grossman said that the trend in 2008 was toward attacks against the browser, while in 2009 we are likely to see a shift back toward servers with HTTP parameter pollution attacks. Also, according to Grossman, as diverse as 2008's Web exploits are, they can likely all be classified according to MITRE's Common Weakness Enumeration (CWE). He gave the example that clickjacking could be classified as a UI redressing attack. Rik Farrow asked if Grossman used NoScript as a countermeasure, and Grossman said that he did. When Farrow then asked the same question of the audience, very few hands went up. (See p. 16 of this issue for Grossman's article on Web hacks and p. 21 for an article on NoScript.)

POSTER SESSION

Posters below summarized by Kalpana Gondi (kgondi@cs.uic.edu)

- ***An Examination of Secure Programming Practices Through Open Source Vulnerability Patch Characteristics***
  Mark Plemmons, Andrew Falivene, Jonathan Peterson, Adam Wenner, Will Quinley, Jing Xie, and Bill Chu, University of North Carolina at Charlotte

Mark Plemmons presented a study to understand secure programming practices by analyzing characteristics of patches for vulnerabilities in open source applications. Authors analyzed Linux kernel vulnerabilities and their patches for the period of 2006–2008 (from kernel.org) to learn characteristics such as number of files and lines changed. For the buffer overflow vulnerabilities, patches were localized to a small number of files. The methodology followed was software vulnerability cognitive complexity (SVCC), where SVCC = Lines of Code added + Lines of Code removed.

- ***The Impact of Structured Application Development Framework on Web Application Security***
  Heather Lipford, Will Stranathans, Daniel Oakley, Jing Xie, and Bei-Tseng Chu, University of North Carolina at Charlotte

Will Stranathans presented this work about the effects of structured application development frameworks on Web application security. The authors have studied practically, by training a few members to observe the behavior of the software they have written after taking the training in application development frameworks (especially struts). There were two groups of people, those with and without the knowledge of application development framework, who wrote the software. The software was tested against attacks like SQL injection, XSS, and system information leaks, and there was a major difference in defense against XSS and information leaks between the software developed with and without the application framework knowledge. The authors observed that the knowledge of frameworks will help in writing the secure code.

- ***Toward Enabling Secure Web 2.0 Content Sharing Beyond Walled Gardens***
  San-Tsai Sun and Konstantin Beznosov, University of British Columbia

San-Tsai Sun discussed enabling secure content sharing where there are no proper mechanisms in Web 2.0 to provide content sharing among individuals in a controlled manner across content-hosting or application service provider (CSP) boundaries. The design includes OpenID email protocol and RT policy service. OpenID email protocol enables OpenID identity providers to use email as an alternative identity. RT policy provides services for Internet users to organize their role-based trust-management access control policies and for CSPs to make access decisions. The two key concerns are usability and interoperability.

- ***A Self-Certified Signcryption Scheme for Mobile Communications***
  Ki-Eun Shin and Hyoung-Kee Choi, Sungkyunkwan University

Ki-Eun Shin pointed out that securing mobile communications is challenging because communicating entities (i.e., mobile devices) are resource-constrained, and mobile networks restrict Internet access. Hence, cryptosystems developed for mobile communications should be efficient, and overhead associated with the security protocol needs to be minimal. The intervention by the conventional trusted authority should also be minimized because of the restricted access to outside networks. The authors propose a self-certified signcryption scheme to withstand such obstacles in mobile communication.

- ***Developing Security and Privacy Requirements for a Local Area COllaborative Meeting Environment (LACOME)***
  Fahimeh Raja, Kirstie Hawkey, and Kellogg S. Booth, University of British Columbia

Fahimeh Raja explained that they tried to look beyond usability toward security and privacy. The authors have developed software to provide a multi-user platform to support sharing of co-located and collaborative work in the same platform. This is more useful for group discussions in the corporate world or any group, for that matter, discussing and sharing information on a particular topic. The key challenges here include: authentication, client authentication and authorization, and unintended disclosure of private data. They are focusing on performing one-on-one interviews and focus groups to get the end users' privacy and security requirements and to implement security and privacy controls, especially for those in front of a large audience. Finally, they want to test the scheme in real meetings.

Posters below summarized by Prithvi Bisht (bishtspp@yahoo.com)

- ***Comprehensive Redaction for Neurological Imaging***
  Alex Barclay, Laureate Institute for Brain Research; Nakeisha Schimke and John Hale, University of Tulsa

Alex Barclay presented a technique to preserve privacy of patients' data captured in neurological imaging. According to Barclay, researchers often share imaging data for collaboration and research purposes. However, it may compromise the privacy of patients, e.g., by reconstructing the face, time of capturing the image, etc. He presented a technique to preserve privacy by deleting certain data from images and presented a disk level algorithm to achieve this.

- ***Java Data Security Framework (JDSF) and Its Applications: API Design Refinement***
  Serguei A. Mokhov, Concordia University

Serguei A. Mokhov presented a framework to evaluate/compare various implementations of security algorithms in a homogeneous environment, primarily in Java. According to Mokhov, this framework provides interfaces to implementation providers. Further, according to him, such a framework enables evaluation of disparate implementations of an algorithm on dimensions and metrics set by the framework, such as runtime and memory usage.

- ***Towards Investigating User Account Control Practices in Windows Vista***
  Sara Motiee, Kirstie Hawkey, and Konstantin Beznosov, University of British Columbia

Sara Motiee presented a study plan to investigate patterns of usage for administrative or normal user accounts in Windows Vista. Specifically, the planned study aims to discover if users prefer admin/non-admin accounts, what challenges are faced in non-admin accounts, and how these patterns vary across disparate user groups.

- ***Large-scale Multitouch Interactive Network Visualization***
  Cody Pollet, George Louthan, and John Hale, The University of Tulsa

George Louthan presented an approach to graphically represent network traffic in large-scale networks. The goal is to allow a human to understand and reason about the real-time condition of the network. He further discussed plans to integrate a multi-touch screen in such a system to provide collaboration among different users.

- ***FloVis: Flow Visualization System***
  Diana Paterson, Joel Glanfield, Chris Smith, Teryl Taylor, Stephen Brooks, and John McHugh, Dalhousie University; Carrie Gates, CA Labs

Diana Paterson presented an approach to render the network traffic through visual artifacts. A real-time traffic visualization may allow security administrators to more quickly discover patterns of hostile activity and diagnose compromised hosts. It may also be useful in maintenance: e.g., heavy traffic on a few nodes may represent an in-progress denial-of-service attack as well as a heavily loaded subnet.

Posters below summarized by Asia Slowinska (asia.slowinska@gmail.com)

- ***Beatrix: A Malicious Code Analysis Framework***
  Christian Wressnegger

Beatrix is a framework designed to reduce the effort required to build a prototype for a malware detection technique. The framework introduces a plug-in infrastructure that divides the entire analysis process into six disjoint subtasks: e.g., input, extracting, or formatting. This architecture enables utilizing existing components, which lets the system conduct significant parts of the examination of malicious binaries and thus reduces programmers' efforts. Future work includes extending the set of modules available.

- ***SAND: An Architecture for Signature-Based Automatic Network Protocol Detection***
  George Louthan and John Hale, The University of Tulsa

George Louthan targeted the problem of network traffic classification, and proposed a content-aware method based on the actual contents of packets. His solution overcomes the prevalent lack of adherence to standard port numbers, which could result in incomplete traffic identification in port-based network monitors. The general SAND strategy for identifying a stream involves matching a set of string signatures describing a known protocol format. For example, the SSH identifier finds the strings "SSH-" and "CR LF" and takes the string between them to be the protocol version. Future work includes a thorough analysis of the performance and effectiveness of the system.

Posters below summarized by Patrick Wilbur (patrick.wilbur@gmail.com)

- ***Securing the Application Acquisition Chain: Security Concerns & Human Factors of Application and System Acquisition in the Enterprise***
  Eric Goldman, Rochester Institute of Technology

Eric Goldman explained that this work examines to what extent, if any, security is considered in an organization's selection of information technology software and hardware. They focused on small- to medium-sized organizations, which generally lack the resources, time, and experience to adequately address security. He considered both the processes of acquisition in information technology and the psychology of good decision-making, and he concluded that all businesses and individuals, regardless of size, deal with sensitive and valuable data to varying degrees, and that those organizations and individuals that do not keep up with security concerns become easy prey as others advance their security focus, despite those organizations' seemingly small size and value.

- ***Exploring the Human-Behavior Driven Detection Approach in Identifying Outbound Malware Traffic***
  Huijun Xiong, Chih-Cheng Chang, Prateek Malhotra, and Danfeng (Daphne) Yao, Rutgers University

Chih-Cheng Chang noted that outbound malware network traffic is unintended by the user and does not always correlate with a user's actions on the system or the user's intended actions. In this work, both user inputs and outbound traffic are semantically examined to see if a valid correlation exists between actions and outbound traffic. This work assesses the actions a user performs and exposes invalid outbound traffic, which could signify malware traffic that is undesired by the user.

- ***Towards Improving Identity Management Systems Through Heuristic Evaluation***
  Pooya Jaferian, David Botta, Kirstie Hawkey, and Konstantin Beznosov, University of British Columbia

Pooya Jaferian pointed out that good identity management is absolutely critical in an organization's access control framework. However, this work finds that identity management systems can host significant usability problems. The goal of this work is to expose usability problems in identity management that could result in mistakes being made while using or administering those systems, which, in turn, could result in the improper granting of access to resources.

- ***A Spotlight on Security and Privacy for Future Household Robots: Attacks, Lessons, and Framework***
  Tamara Denning, Cynthia Matuszek, Karl Koscher, and Tadayoshi Kohno, University of Washington

In the future, robots and automation will become increasingly ubiquitous in the household. Tamara Denning and Karl Koscher presented this examination of various household robots for security vulnerabilities, which include hard-coded passwords, lack of strong encryption, and other easy-to-avoid weaknesses. Although each individual household robot proved to be fairly innocuous on its own, this work reveals several complex attack vectors spanning multiple, differently purposed (and differently skilled) household robots that could lead to more serious attack payloads (e.g., combining dexterity-rich and vision-rich robots to hold and scan your keys for duplicates to be made by an attacker).

- ***Performance Testing the Vulnerability Response Decision Assistance (VRDA) Framework***
  Art Manion, CERT/Coordination Center; Kazuya Togashi, JPCERT/CC

Art Manion and Kazuya Togashi presented this work about the Vulnerability Response Decision Assistance (VRDA) framework, a decision support system used for predicting an organization's responses to vulnerability report stimuli. In order for the VRDA framework to be an effective tool, it must accurately predict the organization's responses relative to vulnerability reports. This work analyzes errors in VRDA predictions using several techniques (hit rate, off-by-one, mean of squared errors, and mean of the errors) in order to quantitatively evaluate the effectiveness of the VRDA framework. With the help of the Vulnerability Analysis team at the CERT/CC, this work offers a multitude of numerical error assessment results on various tasks for which the VRDA framework can predict responses.

MALWARE DETECTION AND PROTECTION

Summarized by Andres Molina (amolinaf@nsm.umass.edu)

- ***Effective and Efficient Malware Detection at the End Host***
  Clemens Kolbitsch and Paolo Milani Comparetti, Secure Systems Lab, TU Vienna; Christopher Kruegel, University of California, Santa Barbara; Engin Kirda, Institute Eurecom, Sophia Antipolis; Xiaoyong Zhou and XiaoFeng Wang, Indiana University at Bloomington

Clemens Kolbitsch introduced his talk by mentioning the weaknesses of existing malware detection solutions that rely on binary signatures or on the detection of artifacts. Kolbitsch suggests that a better solution to this problem would be to look for patterns of malicious behavior. He claims that a system based on detecting these patterns is harder to obfuscate and is more stable. A major contribution of the presented work is to provide a solution to detect malware that combines the effectiveness of behavior-detecting techniques with the efficiency of previous solutions based on binary signatures.

First, in a detection phase in which the characteristics of the malware's behavior are determined, the malware is executed in a full system emulator called Anubis. During this phase, the system monitors the interaction with the operating system. This observation allows the tool to perform a detailed analysis in order to generate behavior graphs. These detection graphs describe a sequence of required system calls leading to the security-relevant system activity, together with the dependencies to the related previous calls. Finally, the end host is monitored and all the system call activity of unknown executables is logged and matched against the behavior graphs obtained. The speaker described how this would work, using a trojan horse as an example. Clemens described their proposed way of matching behavior graphs using recorded execution semantics and then extracting data propagation and manipulation formulas.

The evaluation of the system covered aspects of the effectiveness and efficiency of the system, showing that the behavior detection is fast enough for the end hosts and that the approach is robust against any kind of binary obfuscation, including polymorphism and metamorphism. Furthermore, in some cases the system can detect malware variants that were never seen by the graph generator.

A member of the audience said that it is difficult to distinguish installers from malicious behavior, especially given that trojans seem to be the most common type of malware these days. Kolbitsch pointed out that trojans actually download other malware following very precise patterns, and the system described already detects these trojans. He noted that the authors plan to explore this kind of detection further in future work.

- ***Protecting Confidential Data on Personal Computers with Storage Capsules***
  Kevin Borders, Eric Vander Weele, Billy Lau, and Atul Prakash, University of Michigan

Billy Lau began his talk by noting that many users need to work with sensitive data, such as financial records, while using a PC that is not trusted. Lau proposed a solution to this problem using storage capsules. He started by describing a typical workflow for a solution using TrueCrypt, pointing out that such a solution should not be considered safe, because when the document is open it also becomes available to malware. He said that Trusted Boot, another existing solution, is also not completely appropriate because all software is required to be verified before installation, and verifying documents is even more complex.

The authors' solution consists of dividing the modes of operation into normal and secure modes. In the normal mode, a user would face no restrictions but should perform only non-sensitive operations and would not have any storage protection. In the secure mode, network output would be prevented, but editing sensitive documents would be possible and changes would be encrypted to storage capsules. The speaker said that from the user's perspective the workflow should be very similar to using a solution such as TrueCrypt. The solution with storage capsules is implemented by running a primary virtual machine and a secure virtual machine. The I/O is restricted across different modules between the Secure VM, the Primary VM, and the VMM that handles the physical device drivers, while the system is in secure mode.

Paul Van Oorschot from Carleton University questioned the speaker's claim to offer a system with better usability, given that the authors did not conduct a study to evaluate this claim. Another member of the audience also pointed out that since the primary OS is not trusted and the viewer is run in this system, it may be hard for a user to trust the viewer. Lau and one of the co-authors responded that the viewer is not trusted in the proposed model, and it can be thought of as a malicious client. How would their solution compare to simply rebooting the machine into another more trusted OS, given that entering and exiting the trusted modes takes a couple of minutes? The transition in their system would still be faster than rebooting into another system. Would all covert channels be mitigated to avoid leakage from the Secure VM in this system? The authors did not claim to eliminate all covert channels. Instead, the intention was to eliminate as many as possible at the layer of abstraction provided by the authors. An audience member pointed out that buffer overflows may have to be addressed by this architecture. Lau noted that, while this point should be taken into account, the system is currently implemented in Python, a type-safe language.

- ***Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms***
  Ralf Hund, Thorsten Holz, and Felix C. Freiling, Laboratory for Dependable Distributed Systems, University of Mannheim, Germany

Ralf Hund noted that the kernel has elevated privileges and that not even virtualization solutions allow the detection of malicious programs at lower privileged levels. Hund argued that it is necessary to prevent malicious programs from executing in the first place. The key idea is that current integrity protection mechanisms do not protect against attacks in which the attacker re-uses existing code within the kernel to perform malicious computations. He explained how this procedure can be automated by providing a framework with three core components: a constructor, a compiler, and a loader which can currently be used in 32-bit Windows operative systems running IA-32.

Hund described the instruction sequences that are useful for performing these types of attacks. He also briefly described the designs of the so-called automated gadget construction, the compiler, and the loader. Hund gave a demo of the rootkit implementation, which can be used to traverse the process list and remove a specific process. This rootkit is only 6KB in size, and, as shown in the demo, can be used to eliminate the program Ghost.exe from the Windows Task Manager while the process is still running.

Hund concluded by claiming that the problem is malicious computation, not malicious code, and, therefore, code integrity itself is not enough to provide a secure OS. Future work would include the implementation of the rootkit in other operating systems in order to show its portability and to develop some countermeasures against this sort of attack.

A member of the audience asked if address randomization could prevent this kind of attack. Hund responded that indeed this could be effective in mitigating the attack but not in the way it is currently implemented, not even in Windows Vista, where the randomization is done in user space. Could something be done to the kernel compiler to prevent these attacks? Although possible, this approach would require the recompilation of every single system component.

INVITED TALK

- ***Hash Functions and Their Many Uses in Cryptography***
  Shai Halevi, IBM Research

Summarized by John Brattin (jbrattin@student.umass.edu)

Shai Halevi divided his talk on cryptographic hash functions into four main parts. The first described some standard uses for hash functions; the second explained a motivating application for hash functions; the third described how hash functions should be used; and the last described some of the considerations taken into account during the implementation of Fugue, the IBM submission for NIST's SHA-3 hash function competition.

Traditionally, hash functions have been used to compress, encrypt, and authenticate data, or for error-checking. Hash functions can be used in digital signature algorithms, in message authentication codes, in pseudo-random number generators, and in key derivation functions. Halevi described briefly how one would go about implementing each of these things.

The important properties of a hash function are also the defining characteristics of a "random function": each output is equally (im)probable; collisions (e.g., a and b such that H(a)=H(b)] are hard to find; fixed points [e.g., a such that a=H(a)] are hard to find; and so on. The "random oracle paradigm" is a method of creating a cryptosystem in which you assume you have a random function, design a system around that function, prove that that system would be secure, and then replace the ideal random function with a practical, somewhat nonrandom hash function. In most cases, this paradigm is successful, resulting in secure cryptosystems. When the paradigm fails, it is due to some nonrandom property of the hash function.

However, every hash function has some nonrandom properties which can be exploited, because every hash function is computable. There are degenerate cases of cryptosystems designed using the random oracle paradigm, which are provably secure with a random function but trivially breakable with any hash function. According to Halevi, the best way to minimize the effect of the nonrandom properties of hash functions is to rely on very weak security assumptions, i.e., to claim that a cryptosystem is secure only when several external conditions are met. Halevi described several systems with weak security assumptions, including enhanced target collision resistance (eTCR).

The final section of the talk concerned Fugue, IBM's submission for the NIST hash competition. Many modern hash functions rely on the Merkle-Damgard construction, which is an iterative method that hashes the first part of the message, then hashes that result with the next part of the message, and so on. However, this construction has very little intermediate state. At each iteration, a nontrivial amount of work is done to hash part of the message, but that work is all thrown away and only the hashed result is preserved for the next iteration. Alternatively, Fugue carries along 120 bytes of intermediate state, making it harder to find internal collisions, which have been used in attacks on previous hash functions, eventually leading to real collisions.

A member of the audience asked if a PRG could be used to extend a short salt into a longer salt in the eTCR scheme. Halevi replied that it could be done but that the resulting salt may be easier to guess—if the attacker knows the PRG and can guess the seed, then they've generated your salt and it's easier to break the eTCR. Dan Boneh noted that Halevi talked about Fugue's collision resistance but didn't mention other nice properties of random functions—fixed points, low Hamming weight, and so on. Halevi responded that second-preimage analysis worked out to be very similar to the analysis he gave, and pseudo-randomness would also work out to be similar in some ways. He hadn't looked into analysis of fixed points.

BROWSER SECURITY

Summarized by Todd Deshane (deshantm@clarkson.edu)

- ***Crying Wolf: An Empirical Study of SSL Warning Effectiveness***
  Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor, Carnegie Mellon University

In the talk, Joshua Sunshine described a user study done at Carnegie Mellon University (CMU), in which the goal was to study the effectiveness of SSL warnings presented to users by various browsers and also some custom warnings devised by their research group. Most, if not all, modern browsers warn about SSL certificate problems in the cases of domain mismatch, unknown certificate authority, or expiration. These warnings happen in two types of cases, either as a final protection against man-in-the-middle attacks or when a user is contacting a legitimate server. One example of a legitimate server that would issue an SSL warning is the Carnegie Mellon Library Web site, which uses the Carnegie Mellon certificate authority. Since most browsers don't have the Carnegie Mellon authority added by default, the users will see an SSL warning.

Josh showed three native browser warnings, from Firefox 2 (FF2), Internet Explorer 7 (IE7), and Firefox 3 (FF3), as well as two custom warnings, a single-page and a multi-page warning. He noted the change from popup style warnings (pre-2007) to warnings that take up the entire page (post-2007). He also described the shift from a default action of ignoring the warning and continuing anyway to browsers that make it very difficult to ignore the warning, often forcing the user to make more than a single click to get by. In the FF3 case, four steps are necessary to ignore the warning and get through. In fact, in one of the alpha releases of FF3 there were 11 steps.

Next, Josh described his team's principled approach to designing two custom warnings based on an online survey and some warning science guidelines. The outcomes of the online survey taught them that content sensitivity (what the user is currently doing) is important to take into account. Also, they learned that users will ignore warnings out of habit (they ignored the warnings in the past and nothing bad happened). From the survey, they also found that people confused the SSL warnings with much less serious warnings, such as sending unencrypted information over the Internet via a Google search. The warning science guidelines they used, as taken from applied psychology literature, were to avoid warnings when possible, clearly explain the risk, and provide straightforward instructions for avoiding the risk. In their custom multi-page warning, they first asked the user what type of Web site they were visiting (bank, e-commerce, other, don't know). If the user chose bank or e-commerce, then a second page appeared wgucg gave a severe warning, based on the FF3 phishing warning and using the most severe, red Larry icon. The only two buttons on that second page were "Get me out of here" and "Why was this blocked?" with a small link in the bottom right to ignore the warning. If the user chose "other" from the previous warning page (as the type of site), they bypassed the severe warning page and were directed right to the destination site. Their custom single-page warning, as opposed to the multi-page warning, was simply the red, severe warning page, without asking users the type of site they were visiting.

The user study had 100 participants, all CMU students, who were randomly assigned to the different warnings (FF2, FF3, IE7, custom single-page, and custom multi-page). SSL warnings were shown to the user in two of the four tasks, the bank task and the library task. For each task, the users were presented with an alternative method for completing the task, such as calling or using a different Web site. The bank task required them to enter their credentials for the bank, while the library task didn't require any sensitive information. The hypotheses of the study were that the IE7 and FF2 warnings would be ignored at both Web sites, participants would likely obey the FF3 and the custom single-page warning on both sites, and participants who saw the multi-page warning would obey on the bank Web site but continue on the library site.

The hypotheses turned out to be validated, but with some interesting findings along the way. First, although it was less likely that users would continue to the bank Web site, even in the best case (the single-page warning), 45% of the users visiting the bank Web site still ignored the warning! Also, the FF3 warning turned out to be a significant deterrent, even in the library task. Finally, it was surprising that even though fewer people continued through the bank task than the library task, the difference was slight, generally only one or two users. Based on an exit survey, the authors found that their single-page warning was more effective at conveying the risk and intent of the warning. They also noticed that 4 out of the 20 users of FF3 confused the warning with the "Page not found" (404) error. The FF3 warning also caused the most (10 times as much) hesitation actions (clicking back, refreshing, re-typing URLs, etc.) than any other warning.

The CMU team feels there are weaknesses that need to be addressed with their multi-page warning, such as better context-sensitive information. Finally, they conclude that forcing users to do the right thing (such as using systems like Perspectives and ForceHTTPS) is the correct way to go.

Someone asked about the backgrounds of the participants. Josh responded that as students at CMU they were rather technical, many of them in technical master's degree programs, such as information networking and software engineering. What is the general applicability of the results? The results should be applicable to the general population; technical expertise did not seem to play a role in the actions of the users. What is the statistical significance of the graph comparing users who ignored warnings on the library task vs. those who ignored warnings on the bank task? The only two cases where there was statistical significance was for the single and multi-page warnings. Another questioner wondered about the recommendation to force safe practices and in turn get rid of self-signed certificates. Josh said that in an ideal world he would recommend getting rid of self-signed certificates and use Perspectives and ForceHTTPS, although this avenue may be too costly, and he surmised that IE developers would not support it. Had the users entered their actual credentials, which would imply that they could have captured 45% of users' actual credentials? Josh confirmed both were true. Did users have an alternate way to succeed in the task? Josh said that they were allowed to use the phone or visit a different Web site.

- ***The Multi-Principal OS Construction of the Gazelle Web Browser***
  Helen J. Wang, Microsoft Research; Chris Grier, University of Illinois at Urbana-Champaign; Alex Moshchuk, University of Washington; Samuel T. King, University of Illinois at Urbana-Champaign; Piali Choudhury and Herman Venter, Microsoft Research

Alex Moshchuk acknowledged the trend of users to store information in the cloud as opposed to on their own computers. He explained that the existing browser technologies, such as IE8 and Google Chrome, take the approach of protecting valuables on the desktop via sandboxing and other similar techniques, but fail to protect Web sites from stealing data from each other. Therefore, the authors set out to design their Gazelle browser to apply operating system concepts directly in the browser's security model: for instance, moving all of the complicated resource allocation, protection, etc., into a small, trusted, and simple browser kernel. Taking the OS analogy one step further, instead of treating users as principals they treat individual Web sites as the principals by putting them into protection domains. The real challenge arises when dealing with Web sites that embed content from other Web sites, as is done with mash-ups.

The design of Gazelle builds on the concept of same-origin policy by labeling content based on origin and isolating Web site origins into Web site principals. Further, they separate principals into principal instances if, for example, multiple browser tabs from the same origin are open. In their design, principal instances would run in their own processes, but finer-grained methods of separation (e.g., based on type-safe code) could be applied. The architecture of the system is a single browser kernel that mediates all resource access of the principal instances, including network, storage, and user display. Gazelle requires that principal instances use the Gazelle API to interact with the system and each other. Even Flash (and other plug-ins) would run in its own process and would interact via Gazelle function calls.

The Google Chrome and IE8 browsers combine content from different origins on the same tab into the same process, where a malicious Web site could compromise data from another in the same tab. In contrast, Gazelle separates each of the principals from different origins into its own process, thus protecting the data of another site even if one of the principals is compromised. The goals of Chrome and IE8 are reliability and stability (keeping the browser going even if another tab crashes), while the goal of Gazelle is to introduce more security by protecting principals from each other.

Next, Alex briefly discussed the backward compatibility vs. security trade-off and noted that it is a policy issue. He also discussed in detail the concept of display in Gazelle and explained how to use display access control to limit both the creator window (landlord) and embedded content (tenant) in order to allow proper control of data or display. The concepts exist to some extent in modern browsers, but are intermixed too much and the complexity leads to bugs. Gazelle is designed to more easily protect against CSS history stealing and clickjacking by carefully processing events and maintaining the proper protection between principals in the browser kernel.

The implementation is in C# and makes use of the Trident rendering engine of IE. The evaluation described in the talk showed that Gazelle could perform on par with other modern browsers, specifically IE7 and Chrome, for single-origin pages and has a fair amount of overhead for multiple-origin pages. He noted, however, that Gazelle is still a research prototype and has not been tuned for performance. He noted several optimizations that could bring performance closer to Chrome and IE7, and he mentioned Xax and Native Client (NaCl) as examples.

The first two questions concerned the overhead and the feasibility of using Xax and NaCl. Alex responded that he didn't have numbers for the overhead involved, but he didn't think it would be a problem, based on the overhead of system calls they had been experiencing. He mentioned that they are considering implementing a new renderer for Gazelle so that they can take advantage of Xax or NaCl-style sandboxing. He considered the process of adding sandboxing support to be more of an engineering effort, since XaX, NaCl, and Google Chrome had already accomplished it. There was a question regarding plug-ins; Alex responded that they are prioritizing support for a Flash plug-in. Rik Farrow asked about their decision to support transparent frames (thus opening up the possibility for clickjacking). Alex explained that they are studying the top sites and that they considered backward compatibility issues. He noted that the policy of allowing same-origin transparent frames was just one possibility and that other policies are being considered by his team.

INVITED TALK

- ***DNS Security: Lessons Learned and The Road Ahead***
  David Dagon, Georgia Institute of Technology

  Summarized by Michalis Polychronakis (mikepo@ics.forth.gr)

During 2008, the DNS infrastructure underwent a major disruption due to a new and improved DNS cache poisoning attack. David Dagon presented a timeline of the events that led to the public release of the attack details, including the response steps the DNS community took to mitigate the attack, and discussed current and emerging DNS security issues that remain open.

In a poisoning attack, a recursive DNS server is forced to perform a lookup for which it does not have any cached answer and thus needs to ask an authoritative resolver. In the meantime, the attacker floods the recursive server with spoofed answers using the source address of the authoritative server. If the race between the misleading and the legitimate responses ends in favor of the attacker, then the relevant DNS cache entry will be changed to point to any malicious server the attacker chooses.

In order to succeed, one of the spoofed responses has to match the 16-bit query ID used in the original request. In the past, DNS poisoning was not very effective, because correct answers were cached for a long period, so the window of opportunity for the attacker was very limited. However, in early 2008 Dan Kaminsky disclosed a new poisoning technique that is not affected by the retention of legitimate responses and thus has no "wait" penalty. The technique exploits the fact that name server (NS) locations are communicated through updates included in the DNS responses. The attacker first sends a query for a random child label, such as abcd.example.com, and then floods the server with responses containing a malicious NS update. If the query ID field is not matched, the attacker does not have to wait, but repeats the attack immediately by requesting a different child label, e.g., wxyz.example.com, in essence making the attack only bandwidth-limited.

The response of the DNS community was immediate. Since enhancements like DNSSEC cannot be deployed in a very short period, most vendors chose an opaque mitigation technique that increases the size of the query ID by adding 16 more bits of randomness through source port randomization. With an unprecedented simultaneous effort, most DNS server vendors had released patches before any exploit details were released. However, port randomization is not a perfect solution, since it adds a considerable resource overhead, and in many cases it is negated by NAT devices that do not preserve the source port.

Almost simultaneously with the disclosure of the Kaminsky attack, David Dagon and his team proposed 0x20 encoding, a practical technique that makes DNS queries more resistant to poisoning attacks. DNS-0x20 increases the randomness of the query ID by mixing upper- and lowercase in the domain name in the query. Since almost all DNS servers preserve the mixed-case encoding in their answers, attackers now have to also guess the correct mixed-case encoding.

Port randomization and DNS-0x20 have been widely adopted. However, David presented results of studies which suggest that about 20% of the DNS servers remain vulnerable. Unpatched servers can be exploited by attackers for various malicious purposes, including email message interception.

DNS prefetching is another emerging issue that has started affecting the DNS infrastructure. DNS prefetching aims to improve the responsiveness of client applications, especially Web browsers, by aggressively resolving host names in advance, before the user actually requests them. For example, after loading a page, the browser can resolve the host names of all URLs in the page before the user actually clicks on any of them. In another example, as a person types in the address bar, the browser can predict matching domain names and attempt to resolve them with the hope that the correct response will have arrived before the user actually presses the return key.

The spurious requests made due to DNS prefetching can have a major impact on the DNS infrastructure. For example, while typing a domain name ending in ".com", a request for the ".co" TLD may be sent before the letter "m" is typed, incurring extra overhead to the ".co" TLD servers. Furthermore, DNS prefetching can be abused for spam delivery verification using unique domains seeded in spam messages, or as a DoS attack multiplier, through nonce child labels in the victim authority that are posted in popular forums or spam messages.

DNS is also widely misused by malware. An old technique is to change the resolver or manipulate the "hosts" file of the victim computer to redirect traffic to a malicious server. Recently, malware uses "DNS agility" for botnet command and control operations. For instance, the Conficker worm used 50,000 unique domain names per day, making tracking the actual C&C servers challenging.

Finally, David presented the current state and future of DNSSEC deployment. DNSSEC uses public-key asymmetric cryptography to validate content in a zone, providing message integrity. This enables other innovative uses besides DNS mappings, such as distribution of PGP keys and email addresses.

In the Q&A session, Niels Provos asked about the actual severity of the DNS poisoning problem. David mentioned that the remaining unpatched servers are being exploited, but it is hard to actually detect an attack since the observer should be on path with the poisoning in order to detect it. However, the cost for mounting a successful HTTP redirection attack is significant, since it requires additional steps besides attacking a DNS server.

Bill Cheswick mentioned that another similar case of massive cooperation among vendors to fix a severe problem was the SYN packet attack in 1996. (See the article on p. 35 in this issue.)

WORK-IN-PROGRESS REPORTS (WIPS)

WiPs below summarized by Kalpana Gondi (kgondi@cs.uic.edu)

- ***Full-Datapath Secure Deletion***
  Sarah M. Diesburg, Christopher R. Meyers, and An-I Andy Wang, Florida State University

Sarah Diesburg presented this work focusing on erasing file data securely and completely. There exist previous solutions to secure delete, but these cannot guarantee fine-grained deletion. Diesburg is trying to come up with an extension to secure deletion solutions using a module for the Linux kernel to keep track of blocks to be deleted. The proposed approach is to delete not only the files but also the metadata of the files, so that no traces are left in the hard disk. Also, the module takes care of deleting the data residing at multiple locations (e.g., on flash drives). Complete removal is necessary for users' privacy when a large amount of data is to be deleted permanently—individual files from bank accounts, for example.

- ***Cacophony: BitTorrent over VoIP***
  Rhandi Martin and Angelos Stavrou, George Mason University

Rhandi Martin described Cacophony, a usability tool to enable information sharing when there are legal restrictions. It is like steganography where the information sharing is not noticeable to network traffic analyzers. The authors are proposing this solution especially for BitTorrent traffic, which may be blocked by educational institutions or ISPs. The main idea is to hide the BitTorrent traffic in VoIP. As a result this traffic will not be noticeable to the network administrators/packet inspectors who can watch the traffic to block any data. As the authors acknowledge, the main limitation would be to conceal the large amount of BitTorrent traffic given the number of connections. To resolve the size limitation, authors propose the use of multiple vectors such as teleconferencing traffic and relaying.

WiPs below summarized by Prithvi Bisht (bishtspp@yahoo.com)

- ***Easier Passwords for the iPhone***
  Bill Cheswick, AT&T Research

Bill Cheswick presented a system that enables users to quickly and securely enter passwords on the iPhone. Cheswick mentioned that hard passwords are, ironically, hard to remember. This problem is aggravated for mobile phones due to their small form factor, and may make entering such passwords difficult and insecure (e.g., with increased time-to-enter, users can fall prey to shoulder surfing attacks). According to Cheswick, this work attempts to bring "usability" and "security" together. To retain strong entropy of passwords, Cheswick proposed using a combination of multiple, easy to remember, dictionary words as passwords. To allow users to quickly enter the password, Cheswick proposed the iPhone's spell checker feature to reduce the burden on users to enter all words correctly and, hence, reduce the time needed to enter passwords.

- ***Lightweight Information Tracking for Mobile Phones***
  William Enck and Patrick McDaniel, Penn State University; Jaeyeon Jung and Anmol Sheth, Intel Labs Seattle; Byung-gon Chun, Intel Labs Berkeley

William Enck presented an approach to implement information-flow tracking in mobile phones. He briefly touched upon the lack of security in existing mobile phones and presented an approach to enable taint tracking in Android Virtual Machine. According to Enck, mobile platforms are amenable to such taint tracking, as most sensitive information is retrieved from single-purpose interfaces and thereby allows for automatic tagging. He further indicated the possibility of enforcing precise security policies in the presence of system-wide taint information. Preliminary results indicate that the taint tracking does not adversely impact the performance of the system.

WiPs below summarized by Patrick Wilbur (patrick.wilbur@gmail.com)

- ***The OSCKAR Virtualization Security Policy Enforcement Framework***
  Todd Deshane and Patrick F. Wilbur, Clarkson University

User applications and virtual appliances (applications packaged within virtual machines) can be difficult to secure and, upon distribution, can potentially be run in environments that are ill-equipped to meet their unique security requirements. Furthermore, without clear environmental awareness of the specific needs and behaviors of applications and virtual appliances, malware can exploit vulnerabilities and wreak havoc on the installed system and others. This work develops an extensible policy enforcement framework and contract specification—where the application package maintainer who knows best can clearly define the application's needs and behaviors—to mitigate malware problems by eliminating the majority of their payload.

- ***An Introduction to LR-AKE (Leakage-Resilient Authenticated Key Exchange) Project***
  SeongHan Shin and Kazukuni Kobara, Research Center for Information Security (RCIS), National Institute of Advanced Industrial Science and Technology (AIST), Japan

SeongHan Shin described Authenticated Key Exchange (AKE) protocols as a core cryptographic primitive, and these are used for establishing both mutual authentication and secure channels of communication. Traditional AKE protocols assume shared secrets are and will remain secure; in practice, however, these shared secrets are often leaked, resulting in the cryptographic system becoming insecure. The Leakage-Resilient Authenticated Key Exchange Project works to design a new Authenticated Key Exchange method that mitigates the risks associated with the leakage of shared secrets, while also recognizing the threat of dictionary attacks on traditional password-based protection of shared secrets.

- ***Towards Exploitation-Resistant Trust Models for Open Distributed Systems***
  Amirali Salehi-Abari and Tony White, Carleton University

Amirali Salehi-Abari noted the importance of trust models in establishing agent reputation within distributed systems, where the reputation of an agent is formed from information collected from those who have interacted with the agent in the past. This work adds exploitation resistance to the traditional trust model criteria. This requires the trust model to also protect against the exploitation of the system by an adversary who understands the internal workings of the trust model in use. This work thwarts trust exploitation by an individual agent by cautiously incrementing trust after defection and issuing larger punishments after each defection.

- ***Scalable Web Content Attestations***
  Thomas Moyer, Penn State University

As Web content is received, a Web user can tell which server the content is being sent from, but a user generally has no indication of the integrity or authenticity of the content they are viewing. This work attempts to better inform the user of the level of content integrity by leveraging TPM and integrity measurement technologies. This work provides a clear binding between the state of a server hosting content and the content being served, and does so efficiently despite the slowness of conducting TPM operations.

- ***Exploring the Trusted Computing Base of User Applications***
  Hayawardh Vijayakumar, Penn State University

The Trusted Computing Base (TCB) of an operating system is large, consisting of both the kernel and trusted programs, despite the fact that an individual user application might only need to interact with parts of the kernel and some trusted applications. Based on the hypothesis that typical user-space applications use only a small fraction of the TCB that runs on a typical OS, this work looks at applications and attempts to find what that fraction is. This work then deploys an application in separate virtual machines equipped only with the dependencies the application needs.

- ***Further Improving Tor's Transport Layer***
  Chris Alexander, University of Waterloo

Tor's current transport layer employs a user-level TCP stack and Round Robin to transfer data and multiplex data across common paths, which poses problems with resending portions of multiplexed data as well as fair congestion control in accordance with the primary purposes of Tor. This work replaces the user-level TCP stack with user-level Stream Control Transmission Protocol (SCTP) in order to decrease memory requirements and allow customization of the congestion control mechanism. Furthermore, this work replaces the round-robin scheduling so that bursty traffic (e.g., HTTP) can compete with large, steady traffic (e.g., BitTorrent), which is fairer for traffic closer to Tor's primary purposes (i.e., anonymity and censorship circumvention).

WiPs below summarized by Asia Slowinska (asia.slowinska@gmail.com)

- ***Challenges in Sinkholing a Resilient P2P Botnet (Is It Possible or Not?)***
  Greg Sinclair, iDefense/UNCC; Brent Hoon Kang, UNCC

Brent Hoon Kang presented the layered architecture of Waledac, a P2P botnet, and described the resilient protection mechanisms that Waledac has employed to protect the botnet against common mitigation efforts. The hierarchy introduces a number of layers, starting from spammer nodes at the bottom. Above these come repeater nodes, which forward messages to/from higher parts of the hierarchy. Subsequently, the TSL layer protects the bot master located at the top. The research conducted demonstrates that Waledac introduces a number of mechanisms preventing it from being sinkholed. For example, taking down the TSL layer nodes may result in converting some of the repeater (or new) nodes to new TSL nodes. Also, unlike the Storm botnet in the past, the new information about new TSL server lists is signed by the private key of the upper layer, making any modification to TSL information impossible.

- ***Advanced Metering Infrastructure Security Analysis***
  Steve McLaughlin and Patrick McDaniel, Penn State University

The Advanced Metering Infrastructure (AMI) is a set of smart meters and communication networks forming the basis of smart grid. These smart meters observe one's energy consumption and communicate that information to customers, the local utility, and the grid. Steve McLaughlin presented his research on the security implications of the meter functionality. The penetration testing that was conducted employs an attack tree methodology, where the root of the tree defines a goal, e.g., committing energy theft. The subgoals are defined as the child nodes. Finally, the leaves determine the types of attack to mount. The results identify numerous vulnerabilities and exploits possible: by unplugging a meter from a phone connection at the right time, for example, one will be able to impersonate the meter using a regular computer and forge demand values. The work is published at CRITIS 2009.

- ***Enhancing the Formal Cyberforensic Approach with Observation Modeling with Credibility Factors and Mathematical Theory of Evidence***
  Serguei A. Mokhov, Concordia University

Serguei Mokhov presented his work on refining the formal cyberforensics approach by Gladyshev to model cybercrime investigations, evidence, and witness accounts in order to reconstruct the event and verify whether a given claim agrees with the evidence. With the invention and use of an intensional programming language called Forensic Lucid, he improves Gladyshev's finite state automata (FSA) approach in order to increase the usability of the entire system. Current work is being conducted to enable defining credibility of witnesses or evidence in a case using the Dempster-Shafer mathematical theory of evidence.

- ***Decompiling Android Applications***
  Damien Octeau, Penn State University

Damien Octeau presented his approach to decompiling the Android application bytecode back to Java source code. He claimed that this might prove useful for several reasons: for instance, to unearth security policies buried in the source code, which is inaccessible if one holds a binary only. Android applications are written in Java but run in a virtual machine called Dalvik VM, which significantly differs from the traditional JVM (Dalvik VM is register-based, for example, as opposed to the stack-based JVM). Octeau built a tool for converting Dalvik executable files (.dex) into new Java bytecode files (.class), which can be further processed by existing Java bytecode tools to recapture the original source code. Initial results show that the method is effective.

CSET '09: 2nd Workshop on Cyber Security Experimentation and Test

Montreal, Canada
August 10, 2009

Sessions below summarized by Arun Viswanathan (aviswana@usc.edu)

OPENING REMARKS

Douglas Maughan, Program Manager in the Cyber Security R&D center from DHS, opened the conference on behalf of General Chair Terry Benzel from USC/ISI by giving a very brief talk on the importance of testbeds and security experimentation.

He was followed by Jelena Mirkovic from USC/ISI and Angelos Stavrou from George Mason University, who welcomed the attendees and thanked the Program Committee members. Jelena presented the statistics for CSET '09. There were 27 papers submitted for the conference, of which nine were accepted. Three papers were off-topic and were rejected. Of the 22 finally reviewed, 13 were on experimentation (four were accepted), five on testbeds (three accepted) and four on education (two accepted). She noted that the common problems found in rejected papers were lack of novelty, bad timing, and missing lessons learned.

On the future of CSET, she was enthusiastic that total submissions were up this year, with 25/27 papers coming from people unrelated to the DETER testbed. She, along with Angelos, commented on the lack of awareness among researchers about existing testbeds such as DETER/GENI. This situation will, hopefully, improve with newer and larger testbeds like GENI and NCR. Jelena noted a need for more submissions in the areas of education, tools, experiment methodology, and result validation. She concluded by stating that she was hopeful about having CSET '10 co-located with USENIX Security '10.

KEYNOTE ADDRESS

- ***The Future of Cyber Security Experimentation and Test***
  Michael VanPutte, DARPA Program Manager for US NCR

Michael VanPutte started his keynote by giving a short tour of the DARPA mission and key accomplishments from 1960 to date (from contributions during the space era, through their key role in building the Internet, to today's latest in warfare). He then classified today's cybertesting communities into two groups: operational and R&D. The cyber-operational community's mission is operational testing and training, whereas the mission for the R&D community is to experiment with new ideas. The operational community deals with inflexible, expensive, special-purpose testbeds, does manual configuration and management, has rigid test schedules, deals with constraining bureaucratic policies, and is largely driven by operationally focused policies. This leads to unrealistic testing, questionable results, and slow research-to-operation transition, and it rarely produces production tools. The R&D community, on the other hand, deals with advancing current understanding, generating and testing newer ideas, and managing flexible but potentially unstable systems.

VanPutte talked about the importance of measurement in science in general and cyber research in particular, which is the key reason for the NCR being part of the President's Comprehensive National Cybersecurity Initiative (CNCI) program. The main goal of the NCR is to "provide a realistic and quantifiable assessment of US Cyber research and development technologies to enable a revolution in national Cyber capabilities and accelerate transition of these technologies in support of the CNCI." The NCR will be the measurement capability for cyber research for both civilian and military sectors.

VanPutte then laid out NCR's key challenges: security—securely running multiple tests at multiple security levels; range configuration and management—securely and safely allocating thousands of heterogeneous resources; test configuration and management—using GUIs for configuring and running tests; usability—building recipes for testing, having malware repositories to assist experiments, and having attackers and defenders provided as a service; realism—having 10K nodes along with chip-level heterogeneous VMs; test time—accelerating test time to reduce time to result; scientific measurement—doing forensic data collection, analysis, and presentation of results; and traffic generation—simulating traffic conditions with human behavior.

VanPutte described the program timeline for NCR. The design phase is over and the program is starting the prototype phase. Selected proposals will have 18 months to build a prototype, after which the program will enter the full-scale construction phase. Finally, in closing, VanPutte provided two ways in which everyone could participate in the effort: through government working groups, such as the Security Accreditation Working Group and Joint Working Group, and via upcoming conferences on security metrics, the science of cyber testing, and CONOPS development.

Andy Thompson from JHU asked about the possibility of open sourcing NCR. VanPutte said that it is a possibility but will strongly depend on the transition partner. Roy Maxion from CMU commented that he liked VanPutte's presentation because it clearly compared how things are with how they should be. Jelena Mirkovic from USC/ISI asked if the NCR will develop a workforce for attack technologies. VanPutte responded that the NCR may be used to evaluate the security of systems but will not create attack technologies. Jelena made a comment that the public knowledge base of NCR should have the ability to take inputs from the knowledge bases of already established testbeds. Angelos Stavrou asked how we could achieve diversity in hardware in the testbed. VanPutte acknowledged that it was a hard question but said that people have been experimenting with segmenting testbeds to achieve hardware diversity. Minaxi Gupta from Indiana University commented on the importance of real data sets to understand attacker behavior. She asked about efforts to make available real-time data sets. VanPutte said that real data from real attacks may include operational data and thus are difficult to unclassify. He said he would still need to look into the specifics of this. Ken Zatyko from BBN asked about the usage of NCR and the kinds of tests that would be run on it. VanPutte responded that NCR is primarily meant for large-scale tests for now but it would heavily depend on the transition partner. Steve Schwab from Sparta asked, "How big is big enough" for a testbed? VanPutte said they need to do the math to determine the statistically significant size for specific experiments.

SECURITY EDUCATION

- ***A Highly Immersive Approach to Teaching Reverse Engineering***
  Golden G. Richard III, University of New Orleans

Golden Richard presented his experiences with developing a hands-on reverse engineering course at the University of New Orleans. He described the course focus as being on reverse engineering malware, with an emphasis on understanding the theory of reversing.

An education in reverse engineering is absent from academia because a course in RE could be really hard on instructors, there is a perception that a semester is not enough to teach RE, the university might object to it, and, finally, there is a perception of limited student interest, which turned out to be quite untrue. To overcome these issues, Richard stressed the importance of building trust with the university and the students. He did not have any problems with the university and he laid down the law for student conduct and informed them of the impact of being involved in malicious activities. Students were thus careful and self-policing. Richard's reasons behind teaching reverse engineering were to train students for deep systems research and teach proper ASM/OS skills, apart from the fact that students were begging for such a course and he himself wanted to do it.

His audience for the course, taught for the first time in spring 2009, consisted of 25 students (2/3 graduate and 1/3 undergraduate). About 1/5th of the students had some OS internals knowledge and very few had any serious ASM skills, which proved challenging. The topics covered included the basic importance of RE, ethical and legal issues, techniques/tools used for RE, basic malware background, Intel assembler introduction, Windows PE formats, C basics, common malware functionality (e.g., delta offset calculation, API address discovery), and ended with anti-debugging/anti-VM technology. The lab setup for the course consisted of an isolated gigabit network, with workstations running Linux with Windows XP VMware images running as guests. The XP image consisted of popular tools like OllyDbg, IDA Pro, Sysinternals Suite, HBGary Responder, VC++ , MASM32 SDK, and some industry-grade forensic tools.

Richard's approach to teaching the class was to immerse students in reversing malware samples immediately. Students started with very simple malware like Michelangelo, which required very basic skills, and progressed on to more difficult samples like Harulf and Conficker. Lectures were first given using PowerPoint, then students were given reversing assignments (performed in teams), followed by use of a document camera for assembly code walk-through, followed by lab sessions, and, finally, students producing documented ASM code for the assignments. The exams were based on assignments and were mostly focused on converting malware code to documented assembly. In conclusion, Richard described the course experience as fun, with great student interest and positive feedback. His course will now be offered on a regular basis at UNO.

Someone asked about the schedule of the course. Richard said that it was a 15-week course with two sessions of 80 minutes each per week. Angelos Stavrou asked about the kind of support students were provided in lab. Richard said that there was no real support team in the lab other than the professor. Stephen Schwab asked if the course is teachable using only open source tools, to which Richard said it was possible but that IDA Pro is well worth the cost of licenses. What key challenge did the students face in the course? Lack of assembly skills. Doug Maughan of DHS offered to make HBGary available for the course. What did the students think RE was about when they first went in? Richard answered, "Cool hacker street cred." Someone asked about the audience for the course. Richard said it was a mix of undergrads and grads; he found the undergrads to be more dedicated, whereas grads varied. Could the course be offered online? It would be very difficult, especially because it heavily relied on the document camera.

- ***Collective Views of the NSA/CSS Cyber Defense Exercise on Curricula and Learning Objectives***
  William J. Adams, United States Military Academy; Efstratios Gavas, United States Merchant Marine Academy; Tim Lacey, Air Force Institute of Technology; Sylvain P. Leblanc, Royal Military College of Canada

Efstratios Gavas described their experiences with NSA/CSS Cyber Defense Exercises (CDX) and its effectiveness in teaching information assurance. Gavas started out with an overview of CDX, which is in its ninth year of competition. It is a four-day exercise but typically requires months of preparation. CDX involves a red team vs. blue team competition, with a white team monitoring. Eight teams participated in the exercise (AFIT, NPS, RMC, USAFA, USCGA, USMMA, USMA, USNA), with RMC from Canada participating for the first time. Each team was given a network and a mock budget to secure a poorly configured network. The network is supposed to be fully functional and provide services like email, IM, a Web server, etc., in the presence of live attacks from the NSA red team. The teams were also supposed to deal with exercise "injects" such as forensics, help-desk requests, DNS, and network reconfigs, which are purposely introduced to simulate real-world administrative chores.

Gavas first gave an overview of the USMMA and its preparations for CDX. USMMA has no formal computer science or information assurance program for participating in the CDX. The USMMA also had only five students participating in CDX this year. As preparation for CDX, the team used a number of virus scanners to detect malware in their systems, used a bunch of network and process monitoring tools to detect suspicious activity, rebuilt their Web servers, and used graphical management tools (monowall and eBox) to simplify administration for their network. The team's results for the exercise were mixed.

Next, Gavas shifted to the results of other academies and their experiences with the exercise. He pointed out that differences between participating academies arise because of the different curriculum and learning objectives. USMA participates with a large team of 30–60 students. They have a very security-active CS department with an ACM chapter and a senior-level capstone elective titled "Information Assurance," which form a basis for USMA participation in CDX. As for the CDX experience, USMA cleaned workstations with a homemade Tripwire-like script and rebuilt the DB and Web server without seeing any significant compromises. As for AFIT, Gavas mentioned that they have a very good graduate program, with courses and labs specifically built for CDX training. Their participation was with two teams of 15. For the CDX, AFIT used IPSec effectively, utilized proxy servers, and mitigated compromises with least-user privileges. RMC from Canada participated for the first time in the competition. Details were not provided about their experience in CDX.

Gavas concluded his talk by giving details of the attacks used by the red team. There were 21 significant distinct compromises made; the most effective attack for the red team was malware callbacks, and the most interesting exploit was the OpenFire remote access exploit, which became public only a few days before the exercise. There was no time left for questions.

SECURITY EXPERIMENTATION

- ***Evaluating Security Products with Clinical Trials***
  Anil Somayaji and Yiru Li, Carleton University; Hajime Inoue, ATC-NY; José M. Fernandez, École Polytechnique Montréal; Richard Ford, Florida Institute of Technology

Anil Somayaji presented an alternative method to evaluate security solutions using Security Clinical Trials, which sparked a very lively and interactive session with lots of discussion. Somayaji made two observations: that regular users face a huge challenge in evaluating security products and standard lab-based practices used for evaluating and comparing security products prove very ineffective for users in reality. Standard practices do not account for a lot of real-world variables such as interaction of the product with different software, users, systems, uses, and attack profiles, and thus cannot measure the actual security provided by the product. He proposed the idea of learning from the field of medicine, where they use clinical trials to overcome similar challenges of genetic diversity, environmental diversity, individual history, etc., in identifying effective remedies. Applied to the field of security, the idea proposed is to evaluate security products "in the field" with real users. Questions answered will be of the nature, "Does it work?" rather than "Why?" or "How?"

Their approach will be to isolate variables of interest via sampling and randomization and then measure indicators and outcomes. Somayaji presented a simple example for anti-malware software evaluation, where 1000 customers of a major home ISP are randomly selected. They are given incentives like free tech support and automatic off-site backups to encourage them to participate. Users are then assigned one of three major antivirus programs. A variety of measures are then used to monitor users and computers involved in the study over a period of three years, to learn the effectiveness of the antivirus solutions. Somayaji then discussed objections to this approach: the significant differences between biology and computers, the utility of such an approach, and the expenses involved. Although there were lots of issues with this approach, Somayaji said in conclusion, clinical trials are one way to determine the effectiveness of solutions in practice and complement lab-testing approaches.

Ken Zatyko from BBN asked why they chose to make a comparison with medicine and not with the criminal system. Somayaji said that the medical perspective was for looking at which defenses are the best. Steve Schwab asked about the legal issues arising out of comparing different organizations. Somayaji acknowledged that there was no way this could be done without the support of the organizations being tested, and he talked of some already willing to do this. John McHugh from Dalhousie University pointed out that medical companies participate in trials because of legal requirements, but antivirus vendors may not have any incentive to participate. Somayaji said this was a public policy question. There was also discussion on self-selection biases negating such trials. Somayaji pointed out that they are incentivizing random users to join the study by providing free backups, technical support, etc., to take care of self-selection biases. Angelos Stavrou asked why the ISPs could not do this themselves by monitoring user traffic, their product updates, and incidents. Somayaji said that this would potentially create a large biased sample and thus was a question of experiment design. Someone asked how they measure the outcomes. Somayaji responded that for now their method is to do retrospective analysis on automated low-level backups. Ray Maxion from CMU concluded the Q&A by interjecting that the "audience is inflicting death by a thousand cuts." His point was that they make such stuff work at CMU all the time and hence this should not just be dismissed. His last point to Somayaji was that as they are proposing a methodology, they must compare it with other methodologies.

- ***The Heisenberg Measuring Uncertainty in Lightweight Virtualization Testbeds***
  Quan Jia, Zhaohui Wang, and Angelos Stavrou, George Mason University

Zhaohui Wang started with an overview of the Heisenberg uncertainty principle followed by a brief discussion of the advantages of lightweight virtualization: process-level isolation, no interprocess communication, high efficiency, no requirement for any I/O or device driver virtualization, and only one copy of the OS image required. This work addresses the question of determining the maximum number of OpenVZ containers that could be run on a server.

The testbed architecture consisted of a Dell PowerEdge 1950 server equipped with two QuadCore Intel Xeon 2.66GHz processors, 8GB RAM, and Gigabit Ethernet. The software used was OpenVZ on a vanilla Linux kernel 2.6.24, along with the UnionFS stackable file system to reduce the memory requirements of the system. Each OpenVZ container ran only five processes: init, syslogd, dbus, sshd, and wget. The measurement approach used was to statically determine the shared and non-shared memory pages for each container and then evaluate the runtime CPU and memory consumption of the Virtual Execution Environments (VEEs) by monitoring /proc file system from the host. The experiments consisted of running containers in groups of 100, 200, 400, 600, 800, 1000, 1200, and 1400 containers, with each container running a wget process that would continuously fetch a pages from an Apache server in random intervals varying from 1 to 10 seconds. The monitoring process was run with varying sampling intervals of 0.1, 0.01, 0.005, and 0.001 seconds.

The results from the experiments showed that the completion times for the experiment increased as the number of containers was increased, but there was a profound increase when the frequency of measurements was increased. The conclusion drawn was that the more you measure, the more you lose. Zhaohui claimed that their work unveiled for the first time the uncertainty problem due to system resource contention in a lightweight virtualization environment. He pointed out that it was not a trivial task to determine the maximum number of VEEs that can be run on a physical host, due to this form of Heisenbergian uncertainty.

Roy Maxion from CMU asked why the CPU utilization maxed out at 600 VEEs for 0.1sec frequency in Figure 4? Zhaohui answered that the contention between the containers caused them to reach a threshold. As for the graphs at other frequencies, they were already affected because of over-measuring.

- ***The Virtual Power System Testbed and Inter-Testbed Integration***
  David C. Bergman, Dong Jin, David M. Nicol, and Tim Yardley, University of Illinois at Urbana-Champaign

Tim Yardley from UIUC presented their work on the Virtual Power System Testbed (VPST), which is a part of the larger Trustworthy Cyber Infrastructure for Power Grid (TCIP) project. TCIP works on securing devices, communication, and data systems that make up the power grid. VPST is designed to support exploration of security technologies being developed for large-scale power grid infrastructure. VPST at the core consists of RINSE, which is a network analyzer and simulator. RINSE is capable of performing high-performance, high-capability network analysis along with multi-resolution modeling of traffic and topology. VPST itself can be connected via secure links to external testbeds and utility power stations.

Yardley mentioned that SCADA systems prompted the work on VPST. SCADA research has a high barrier for entry and thus emulation of these systems can alleviate part of this concern by using accurate models. He mentioned that VPST is designed to leverage valuable resources from other testbeds such as DETER. Yardley then described the interconnection requirements of VPST. Secure interconnection between testbeds and between VPST and utility companies is a prime requirement. He mentioned use of Open PCS Security Architecture for Interoperable Design (OPSAID) in their architecture. Next, performance is a key requirement, as it is very important to keep latency low across multiple testbeds. VPST implements look ahead to keep simulation as close to real time as possible. Resource allocation is the next key aspect, and VPST tries to use a decentralized approach where interfaces to other testbeds are decomposed into modules for ease of customization. Reproducibility is important in SCADA systems because the dynamics of real SCADA networks cover a wide range of conditions, such as size of network, type of underlying physical medium, available bandwidth, and time-varying traffic patterns. Reproducibility is complicated due to human interactions with the system. The system must be able to record interactions and replay. Fidelity is the last of the key requirements, which means that VPST must be as transparent as possible to real devices. This also means that access is needed to real-time data patterns from utility companies.

Yardley next described the use cases for VPST. The first use case is in the training and human-in-the-loop event analysis. VPST allows captured system state to be replayed on the testbed, which can help in making better control decisions and rectifying decisions which may have led to failures in real situations. The second use case is for analysis of incremental deployment. As SCADA networks are large and complex, introducing any new technology must be done carefully. VPST can provide an alternate deployment for testing new technology before deploying it directly into real networks. The third use case is in analyzing the robustness of a design against attacks. Yardley concluded his talk by mentioning their future work on developing a black-box implementation of VPST for DETER.

Roy Maxion from CMU asked how they validate their results. Yardley replied that the system is not yet fully implemented and validation issues have not been fully addressed. Yardley also said that connection to real utility company networks is limited by legal constraints. Angelos Stavrou of GMU asked how they validate fidelity of each component in the network. Yardley said it depends on whether they are using models or real devices. For models it depends on the implementation of the model. Roy Maxion asked about the impact of errors introduced in simulation due to modeling proprietary devices. Yardley said that the issue had not yet been addressed.

Sessions below summarized by Eric Eide (eeide@cs.utah.edu)

- ***Dartmouth Internet Security Testbed (DIST): Building a Campus-wide Wireless Testbed***
  Sergey Bratus, David Kotz, Keren Tan, William Taylor, Anna Shubina, and Bennet Vance, Dartmouth College; Michael E. Locasto, George Mason University

Anna Shubina described her group's experiences in developing and deploying the wireless portion of the Dartmouth Internet Security Testbed (DIST). The wireless infrastructure supports experiments that require access to real-world network traffic. The hardware architecture includes 200+ WiFi access points, called "air monitors," distributed over ten buildings at Dartmouth. The air monitors send captured frames to DIST servers, which process the frames. An experiment describes the kinds of frames to be collected at the monitors and the processing steps to be run at the servers.

The software architecture is carefully designed to protect users' privacy and enforce experimenters' accountability. The air monitors discard all but the MAC layer of each captured frame. The frames are encrypted before being sent to the DIST servers; the servers decrypt and anonymize the frames before making them available to an experiment for analysis or storage. Unsanitized data is never written to disk. The testbed enforces accountability by keeping careful audit trails. For example, DIST policy is that an experiment's source code be checked into DIST's revision-control system before it can be deployed.

One of the technical lessons learned was that a long-running testbed in a production environment must be designed to survive unexpected changes. An unannounced change to Dartmouth's network highlighted the need for a fallback control channel to the air monitors. Shubina also described the many lessons learned in obtaining approval to deploy the wireless network at all. The project required extended negotiations with many organizations within Dartmouth, with issues ranging from the system's security architecture to the aesthetics of signage and the deployed hardware.

After the talk, a CSET attendee asked how often the encryption keys are changed at the air monitors. Shubina replied that they are changed for every experiment. In response to another question, Shubina said that their system does not stop collecting data when the number of network users is low; protecting privacy in such situations is a research issue. Finally, someone asked how long it took to solve all the administrative and social deployment issues. Shubina said that it took two years from start to end.

- ***An Emulation of GENI Access Control***
  Soner Sevinc and Larry Peterson, Princeton University; Trevor Jim and Mary Fernández, AT&T Labs Research

GENI is a planned testbed for exploring new network architectures at scale. It is designed as a federated testbed, with resources controlled by multiple administrative domains. As such, the evolving GENI security architecture is designed to support features such as distributed access control. In this talk, Soner Sevinc described an experiment that he and his colleagues performed to evaluate their design of a distributed access-control mechanism for GENI, driven by data collected from an existing large-scale testbed, PlanetLab.

To perform an operation in GENI, an agent must supply a set of cryptographically signed certificates to show that it is authorized. This involves collecting a chain of certificates, from the root GENI authority down, to establish the agent's identity and privileges. Building these chains means obtaining certificates from multiple administrative authorities. Soner and his colleagues designed a system to optimize the process of certificate collection. Their system, based on a framework called CERTDIST, handles both distribution of certificates and the evaluation of security policy. CERTDIST uses a distributed hash table (DHT) to cache certificates, load-balance requests, and provide fault tolerance.

How can this distributed access-control system for GENI be expected to perform in deployment? To answer that question, Soner and his co-authors started by collecting traces of access-control events in PlanetLab. From these traces, they produced equivalent scripts of GENI access-control events, translating from PlanetLab's centralized model onto their new distributed model. Finally, Soner and his colleagues used 550 PlanetLab nodes to carry out the events in the translated traces. Their experiments led to three main conclusions about the behavior of their distributed access-control system. First, the DHT effectively reduces the request load seen by certificate authorities, although the system still experiences minor "flash crowds" when popular certificates expire. Second, for the request load in the emulated traces, the DHT-based system does not reduce the latency of requests. Third, when the request load is increased by a factor of 10, the DHT improves the success rate of queries by balancing the load.

Future work will explore caching and retrieval strategies for certificates, to address the issues revealed by their evaluation. The PlanetLab traces that drove their experiments are publicly available at http://www.planet-lab.org/.

EXPERIMENTATION TOOLS

- ***Payoff Based IDS Evaluation***
  Michael Collins, RedJack, LLC

Michael Collins proposed a new approach for evaluating the efficiency of an intrusion detection system (IDS). The traditional method for evaluating an IDS is to view the system as a binary (yes/no) classifier: its false positive and negative rates measured as functions of the system's discrimination threshold. In contrast, Michael proposed modeling the IDS as if the attacker were aware of its capacities—treating the IDS as a constraint on the attacker's behavior and modeling how the attacker would respond.

The general idea is to model an IDS as a zero-sum game between an attacker and the IDS. The game is played over an "observable attack space" (OAS), which is defined by the set of attributes the IDS is designed to monitor. For example, if an IDS is designed to use flow data only, the OAS would not have attributes based on packet payloads. The OAS covers observations during normal network behavior and observations under network attack. For every point in the OAS, two functions are defined. The first is the payoff function: for an attack that maps to a particular OAS point, what value does the attacker receive? The second function describes detection: for a given OAS point, what is the probability that the IDS will detect the attacker? Given this setup, an attack is a multi-round game in which the attacker moves through the OAS, collecting the payoff values. After each attacker move, the IDS may detect the attacker and take corrective action. This model provides a basis for comparing intrusion detection systems: over a given OAS and period of time, the best IDS is the one that minimizes the attacker's total payoff.

Michael illustrated his IDS evaluation methodology over four games involving node acquisition (bots), network reconnaissance, maintaining a back-channel, and network saturation (DDoS). Using the evaluation methodology, for example, one can evaluate different strategies for a DDoS attacker. The game models presented in the talk were purely synthetic. Michael said that his future work will focus on developing more realistic models, based on real-world behavior.

After the talk, someone asked about models in which apparently "normal" network observations still permit high-payoff attacker behavior. Michael replied that this was an interesting question and a topic for future research into models of real-world observable behaviors and attacks. John McHugh asked whether network defense is not a two-party game but a multi-party game in which most of the players are normal users. Michael said that modeling intrusion detection as a three-party game might be reasonable; normal users might be modeled as a third party or as part of the game rules themselves.

- ***Toward Instrumenting Network Warfare Competitions to Generate Labeled Datasets***
  Benjamin Sangster, T.J. O'Connor, Thomas Cook, Robert Fanelli, Erik Dean, William J. Adams, Chris Morrell, and Gregory Conti, United States Military Academy

The final paper was presented by Benjamin Sangster and T.J. O'Connor, who shared their experience in collecting network traffic data from the 2009 Inter-Service Academy Cyber Defense Exercise (CDX). As described in another CSET talk, the CDX is an annual competition in which military academies defend networks from a National Security Agency (NSA) red team.

By instrumenting the CDX, the USMA team sought to address the lack of useful network-traffic data sets for security research. Most commonly used data sets are dated, artificial, and contain trivial artifacts: they are not representative of modern-day adversaries. In contrast, the CDX and similar network warfare games are designed to reflect the design and concerns of current networks: e.g., modern hardware and software, networks at scale, and threats such as zero-day attacks. The CDX involves human decision-makers as both attackers and defenders. A potential method for producing useful network traces for research, therefore, is to instrument network warfare competitions. This approach could automatically label the collected traffic as red-team (attacker), blue-team (defender), or white-team (ordinary use).

To evaluate this approach, and with the approval of NSA, the USMA team deployed three traffic-collection points during the 2009 CDX. One was placed at the border of the NSA team: it collected both red-team and white-team traffic from NSA. The second was installed on the network connection just inside the USMA team's VPN router, and the third was placed on the central switch of the USMA team's internal network. The second and third sensors therefore witnessed the ingress and egress filtering performed at the perimeter of the USMA network. They observed a mix of red, blue, and white traffic.

Benjamin and T.J. described the strengths and shortcomings of the data that were collected. They observed that the 2009 CDX dataset has a significantly different "personality" from some older DARPA datasets, due to the use of modern tools and the involvement of humans in the exercise. The CDX dataset is thus more representative of modern networks in both of these respects. However, the CDX dataset is limited by the nature of the exercise. It has less diversity and volume than a production network would have, and the dataset only covers a four-day period. It was also difficult to clearly label some traffic, for instance, due to the mixture of NSA red traffic with white "cover traffic." The authors believe that automatic labeling could be improved by collecting additional red-team logs, either automatically or manually.

The network data and other logs collected during the 2009 CDX are publicly available from http://www.itoc.usma.edu/research/dataset/.

PANEL ON SCIENCE OF SECURITY EXPERIMENTATION

Panelists: John McHugh, Dalhousie University; Jennifer Bayuk, Jennifer L. Bayuk LLC; Minaxi Gupta, Indiana University; Roy Maxion, Carnegie Mellon University

The final CSET event was a spirited panel discussion about the challenges in doing scientifically rigorous experiments on security topics. Jelena Mirkovic invited each of the panelists to start by describing his or her most important "hard problems" that stand in the way of scientific approaches to security. Each of these led to a great deal of discussion between the panel members and the audience.

Minaxi Gupta said that her favorite topics deal with access to data, both immediate and long-term. For instance, as a security researcher you may not know who has the data you want—and even if you do, you may not be able to get access to it. If you get the data you need, you may not have the resources needed to store it and analyze it. Finally, there is currently no standard practice for going backwards from publications to the datasets on which the publications are based. Minaxi concluded that the security community needs repositories that make long-term (multi-year) datasets available in real time, both raw datasets and derived data products. Doug Maughan and others at the workshop noted that the DHS PREDICT repository (https://www.predict.org/) is an important step toward making security datasets available to the public. Roy Maxion said that while it may be difficult to provide data to others, it is possible, and he offered a benchmark dataset for keystroke dynamics that accompanies a paper on his Web page (http://www.cs.cmu.edu/~maxion/). The data can be used for many tasks that are typical in intrusion and insider detection.

Jennifer Bayuk claimed that the hard problem is the "community problem." One aspect of this is competition, rather than cooperation, among security researchers: while researchers compete against each other, the attackers continue to advance. Competition over small problems does not help the community solve the actual problems being faced, such as how to make maximum use of existing tools and techniques in defense of common attacks. A second aspect is the lack of a basis for cooperation: problems that lack existing datasets are simply not being addressed. In response, John McHugh noted that datasets require a great deal of metadata in order to be useful. Sergey Bratus also added that recent testbeds, such as Dartmouth's DIST, can help to address the "unannotated dataset" problem by enforcing good practices.

John McHugh said that computer scientists have no excuses for bad science; they simply have bad practices. In general, computer scientists are not properly trained to conduct experimental science. They lack background in statistics, for example, and often do not collect data properly. McHugh gave an example in which an analysis of a large dataset was rendered invalid because the analysis assumed that clocks were synchronized over multiple data collectors. In fact, they were not—for most of the data-collection period.

The missing metadata for the dataset, which would have described how the data collectors were configured and calibrated, made the dataset significantly less valuable for scientific study. Finally, McHugh said that the requirements for funding and publishing are currently in conflict with rigorous science. Jelena Mirkovic suggested that funding agencies understand the need for good science, but the security community as a whole does not.

Roy Maxion said that the panel had not yet talked about what it means to have science in security. Science first requires having a tightly focused question—the hypothesis. Constructing a well-formed hypothesis is in fact a very difficult task, because it so often involves putting structure on an ill-structured problem. Second, science requires repeatability and reproducibility. Repeatability means that a single experimenter can perform a procedure several times and come up with the same result; reproducibility means that those results can be obtained by other investigators. Third, science depends on validity. Maxion asserted, "This is the issue that assails our field the most." Internal validity means that an experiment is logically consistent, and there are no explanations for the results obtained, other than the proposed explanation (e.g., no confounds). External validity means that the results are generalizable to a larger population. Maxion suggested that conference program committees demand better descriptions of experimental methods in submitted work. Anil Somayaji responded that the security community was still several steps away from rigor, because nobody currently builds on another person's work. The unanimous response from the panel was that the time for change has come!

4th USENIX Workshop on Hot Topics in Security (HotSec '09)

Montreal, Canada
August 11, 2009

SOCIAL FACTORS AND MINIMIZING TRUST

Summarized by Tamara Denning (tdenning@cs.washington.edu)

- ***Using Social Factors in Digital Rights Management***
  Bader Ali and Muthucumaru Maheswaran, McGill University

Bader Ali began by summarizing the current anti-piracy efforts and their weaknesses. Such efforts include the digital locking of software and hardware (DRM), legal measures (lawsuits), and reducing the availability of pirated content (content poisoning). Any anti-piracy efforts need to be considered from the perspective of all stakeholders: both the content publishers and the end users. For example, DRM fails both because it is vulnerable to hacking and because it hinders the goals of the end user.

Bader Ali continued by pointing out that part of the prevalence of piracy is due to lack of social stigma associated with pirating content or obtaining pirated content. The idea behind this project, therefore, is to leverage economic incentives and social pressure between friends to cope with digital content piracy. More specifically, the project concept is to have content publishers deliver digital content to local distributors in online social networks (OSNs).

Users form groups in OSNs. Content publishers deliver digital content to local distributors, who then sell the content to users in their groups. End users benefit because they are able to acquire content from local distributors at a reduced price. Local distributors benefit because they receive a percentage of the profit from content sales in their group. Distributors benefit because they are able to monitor the circulation of watermarked content and grade distribution groups based on their piracy rates; content publishers can then refuse to deliver content to groups with high piracy rates. The desired end result would be the reduction of piracy due to social pressure from peers in one's group, since the distributor and the other group members are punished for any content that is leaked from that group.

One audience member asked why this approach is better than having the content publishers watermark every end user's content. Watermarking for every user requires overhead, as does tracking and punishing every pirating user. This system proposes moving the punishment for piracy out of the legal realm and into the social realm—in short, by bringing anti-piracy norms into mainstream society.

- ***FaceTrust: Assessing the Credibility of Online Personas via Social Networks***
  Michael Sirivianos, Duke University; Kyungbaek Kim, University of California, Irvine; Xiaowei Yang, Duke University

Michael Sirivianos presented this workshop paper on producing credible assertions via online social networks (OSNs). The problem addressed by this work is how to gauge the truth of statements made by online personas. For example, when browsing the Web one might not know whether or not to trust that a product reviewer on Amazon is actually a doctor, as he claims he is. Other problem areas include dating Web sites, Craigslist, eBay transactions, OSN introductions, and age-based access controls.

The authors propose supporting relaxed credentials, where an assertion made by a user is bound to the probability that the assertion is true. A user posts his assertions to his profile on his OSN, where his friends can tag them as verified or rejected. The challenge here is that friends can collude and lie together; therefore, the system assigns credibility values to taggers. The authors use the Advogato trust metric [Levien et al., Security '98] and employ taggers' credibility ratings to assign a final credibility score to a user's assertion. Assertion-credibility pairs can be provided to others as a signed value produced by the credential system. If a user wants to provide a credential without revealing his or her identity, the system can use idemix (http://www.zurich.ibm.com/security/idemix/).

Someone asked how the system protects against a generally credible group of friends who lie about one thing—for example, high school students who lie about their age. This is mitigated by the trustworthiness that is assigned based on assertion type, and not based on the aggregate score; however, this situation is still problematic. Might this system foster a false sense of security? The system is only meant to produce relaxed credentials, and thus cannot be completely trusted. Another audience member suggested that users post false assertions as red herrings to help protect their privacy on the OSN. The speaker agreed with this idea and stated that posting red herrings would also help verify tagger credibility.

- ***How to Print a Secret***
  Aleks Essex, University of Ottawa; Jeremy Clark and Urs Hengartner, University of Waterloo; Carlisle Adams, University of Ottawa

Aleks Essex presented this workshop paper on how to print a human-readable secret without allowing the participating printers to reconstruct the secret. An example of this technology is cryptographic paper-based voting, where the user marks a ballot with a particular pen to reveal confirmation codes. The work involved three components: oblivious transfer, visual cryptography, and invisible ink.

Oblivious transfer is used as an alternative to a trusted dealer, and would be used to blindly divide a ballot print job between entities. Visual cryptography is used so that the end user can re-assemble the secret—the voting confirmation codes. This work also involves developing an invisible ink that color-matches the non-reactive ink that occludes the confirmation code.

An audience member asked what prevented the second printing entity from printing a ballot, revealing its secrets, and then reprinting the ballot with modifications? There are new, cheap ways to incorporate items into paper and then perform an authenticity check with a scanner. Another audience member asked if the invisible ink was indistinguishable from the visible ink under a microscope, since the authors had specified that it is indistinguishable with a black light. The authors had not yet explored that possibility.

NETWORKS AND SOFTWARE

Summarized by Akshay Dua (dakshay@gmail.com)

- ***MitiBox: Camouflage and Deception for Network Scan Mitigation***
  Erwan Le Malécot, Kyushu University and Institute of Systems, Information Technologies and Nanotechnologies

Erwan Le Malécot presented a new approach for network scan mitigation using MitiBox, a camouflage and deception system. He argued that with the growth of the Internet it has become increasingly profitable for attackers to compromise network-connected devices. Attackers use automated tools that can quickly scan large portions of the network and discover potentially interesting targets (devices with open services). This unwanted network scanning activity now accounts for a significant portion of the traffic, and Le Malécot wants to find an effective method for system administrators to deal with it.

Le Malécot claims that little seems to be being done to fight unsolicited network scans. The predominant approach is to rely on network intrusion detection systems, but using them for accurate and early detection is problematic. Le Malécot proposed a new direction which focuses on reducing the pertinence of information that is "leaked" by a network in response to scanning probes. This can be done by making the network behave uniformly, that is, the network responds in an indistinguishable fashion no matter what traffic it receives. To do so, the system proposed by the authors implements the following processing: (1) drop all malformed traffic, and (2) either drop or reply to traffic with equal probability, replies being forged as necessary. Concurrently, observed traffic sources are assigned a trust level based on their initial behavior. This level is then dynamically updated over time.

One person pointed out that any botnet member could initially behave in a way that makes it trusted and then perform the scanning activity. Le Malécot replied that the trusted status lasts only for a single connection and is therefore temporary. Another person stated that botnets have cheap resources and so they could use multiple resources to gain the system's trust. Le Malécot replied that an attacker with extensive resources at its disposal could indeed bypass certain mechanisms of the system but not all (e.g., he would still need to differentiate between forged replies and replies from authentic hosts).

- ***SPAN: A Unified Framework and Toolkit for Querying Heterogeneous Access Policies***
  Swati Gupta, Indian Institute of Technology, Delhi; Kristen LeFevre and Atul Prakash, University of Michigan, Ann Arbor

Swati Gupta presented SPAN, a system that can unify multiple heterogeneous security policies and allow them to be queried later on. Swati pointed out that when security policies from different domains interact with each other frequently, policy loopholes get created even when each individual policy is configured correctly. For example, the SSH service is configured to run on port 22, but the firewall is not made aware of that fact. Swati also mentioned that most tools today don't deal with multiple heterogeneous security policies. Thus, policy unification is left to the system administrator, who then does it in an ad hoc fashion. SPAN helps to alleviate configuration issues by automatically unifying policies from different domains: e.g., Firewall, SSH, NFS.

SPAN takes as input multiple native security policies, parses them, and stores them in an internal format based on Binary Decision Diagrams. These decision diagrams can handle ranges and make them more suitable to policies with large domains such as firewalls. The unified policies can then be queried using an SQL-like language that also includes special statements, called "constraints," to model policy boundaries.

Someone asked, "Can you feed configuration files as is?" If SPAN supports the application, then its configuration file can be input as is. Can others in the community get involved? Swati was happy to work with them and figure out which other policies to include. Can SPAN scale to SELinux policy files? The current version of SPAN is written in Python and designed for functionality rather than speed. They will look into scalability in the future.

- ***Pre-Patched Software***
  Jianing Guo, Jun Yuan, and Rob Johnson, Stony Brook University

Jianing Guo spoke about patching software before it is released rather than after. She pointed out that patches were slow and error-prone, exposing the user to "zero-day exploits." On the other hand, including runtime checks in software involved high overhead and resulted in compatibility issues.

Jianing's solution was to "pre-patch" software by including latent runtime checks that are enabled selectively in the future. She argued that a significant benefit of this approach is that it provides an immediate response to discovered vulnerabilities without the user incurring any visible overhead until the vulnerability is discovered. Jianing presented a Memsafe prototype that included latent checks for bounds violations. A performance evaluation of the prototype indicated a 10% increase in execution time with all checks off (the default case), a 33% increase with one check on, and a twelvefold increase with all checks on.

One person asked Jianing how she differentiated her work from Valkyrie. Valkyrie used program binaries without any knowledge of the program and was very resource intensive. How can a user discover which particular check to turn on? One way would be to turn all checks on and then see which one fails; another way would be for the developer to work with the user to figure this out. Wouldn't this encourage vendors to write sloppy code? The checks were there only to help increase the quality of code. Does this method catch all bugs left over after static analysis of the program code? Their method guarantees memory safety and not type safety, but they haven't written a proof for that yet.

MOBILE AND THE USER

Summarized by Michael Sirivianos
michael.sirivianos@gmail.com)

- ***Authentication Technologies for the Blind or Visually Impaired***
  Nitesh Saxena, Polytechnic Institute of New York University; James H. Watt, University of Connecticut

Since security often relies on users taking relatively difficult actions, choosing hard-to-guess passwords or timely installation of security patches, a disabled person may not be able to appropriately deal with security tasks. Attackers have actually taken advantage of the vulnerability of visually impaired persons by attacking JAWS, software for screen reading.

Rob Johnson presented this work covering current directions on user authentication for the blind and visually impaired. The first is an observation-resilient user authentication method that relies on a challenge transmitted over an audio headset and on the user performing mod 10 computation. In summary, the method encrypts a PIN with an audio challenge modulo 10 from a terminal, e.g., an ATM. The method itself has some open research issues, particularly the possibility of eavesdropping on the audio channel and the user-friendliness of mod 10 computations. Next, Johnson presented strong password management using a mobile phone. Under this family of methods, a user logs in with his cell, the terminal sends a challenge, and the cell responds by vibrating the response to the accelometer of the terminal. An open issue is to investigate the secrecy properties of vibration channel.

With respect to secure device pairing, the talk focused on the Fake-Audio attack. Blind users may be disadvantaged under such attacks because they will not be able to see the attacker. An observation is that Button-Enabled Device Authentication could protect the user because it replaces sound with vibration. The Seeing Is Believing method is also not appropriate, because the visually impaired would have trouble aiming a camera. Any pairing method that relies only on sound is susceptible to the Fake-Audio attack. The talk concluded that the only appropriate solutions are the vibration-button and vibration-vibration pairing methods.

Someone asked how realistic the audio impersonation attacks are. Rob replied that Nitesh Saxena (one of the authors) and his team are currently investigating the practicality of those attacks. Since the visually impaired usually have acute hearing, they may be able to detect the Fake-Audio attack. Could MadLibs alleviate this? Srdjan Čăpkun commented that Fake-Audio attacks target devices as well as humans.

- ***Towards Trustworthy Participatory Sensing***
  Akshay Dua, Nirupama Bulusu, and Wu-chang Feng, Portland State University; Wen Hu, CSIRO ICT Centre, Australia

Akshay Dua presented his work on trustworthy participatory sensing. Traditional sensor networks have a high hardware cost of deployment. On the other hand, participatory sensing leverages user devices as sensors. For example, GPS sensors in cars can assist with predicting or detecting congestion. However, the very openness of participatory sensing makes them open to abuse. Privacy is also a concern, as users may transmit sensitive information. The first part of Dua's talk focused on abuses with respect to content integrity; how to ensure that a reported event is not a fabrication. Previous solutions to this problem employed reputation and incentive mechanisms. Dua argued that their proposed trusted sensing peripheral (TSP) is a more appropriate solution, since any data that originates from the TSP is considered trusted.

The TSP uses a Trusted Platform Module (TPM) which offers platform attestation, i.e., sensors process the data as expected. It also offers data attestation in the form of origin authentication and verifiable data integrity. The user assigns tasks to the peripheral, and the TSP periodically responds with attested readings. The security problems that the TSP has addressed are: (1) data poisoning, since sensed data are signed by a TPM; (2) spoofing, since a burned-in private key makes it impossible to fake the origin of data; (3) collusion; and (4) the Sybil attack, since the embedded private key makes it impossible to separate identity from the device itself. There are still ways that the system could be attacked: (1) fake events—e.g., a lit candle to fake high temperature; (2) damaged sensors; and (3) effective attacks on the trusted module.

The second part of the talk focused on content protection. One possible solution would be for the sensor to encrypt the data for each individual consumer of the sensor data. Dua argued that broadcast encryption (BE) is more suitable. The TSP can also assist in content protection by providing tamper-resistant key storage, and in the future the TSP may also be able to perform BE. Their group has designed and implemented BE on Nokia N800. BE on the N800 takes only a few seconds; they also think BE with symmetric cryptography will improve performance.

Srdjan Căpkun wondered whether there are scenarios where the attacker can fake events. Dua replied that if they use reputations and a peer-review system they may be able to detect event fakers. Is the Flec OS, which was used in this study, open source? It is not, and to acquire it one has to contact its authors.

- ***Implicit Authentication for Mobile Devices***
  Markus Jakobsson, Elaine Shi, Philippe Golle, and Richard Chow, Palo Alto Research Center

Philippe Golle listed current trends in authentication and mentioned that we are now reaching the limits of password authentication, resulting in two-factor authentication becoming more commonplace. At the same time, there has been substantial growth in the number of mobile Internet devices, which are now used to access personal, financial, and medical data. Philippe and his team performed a user study that showed that device passwords are weak; 50% of users mistype the passwords, and most users report that typing passwords on a mobile device is much harder than on standard keyboards. Users consider having to enter passwords as a more significant annoyance than the small screens of their devices.

Golle noted that proxy solutions and single sign-on solutions do not solve the problem, because they do not identify the user but only the device. Consequently, they do not account for the case of theft. Graphical passwords may have higher entropy and be more memorable but have not caught on. Biometrics have been hampered by high error rates. Golle presented their proposed solution: implicit authentication (IA). IA relies on authenticating users based on their habits and the usage patterns of their devices. For example, if a user arrives at work in the morning, the GPS says that he is in the usual location, he gets a call from his spouse and a message from his boss, he may not need to authenticate his cell phone to the bank again. Their initial steps on implicit authentication consider the following types of data: (a) location and co-location; (b) application usage; (c) biometric measurements; and (d) contextual data such as time of day, calendar entries.

The system computes an authentication score on the device. Scores computed on the device protect user privacy but do not defend against theft or corruption of device. Alternatively, the authentication score can be computed by the carrier, which is more secure but raises privacy concerns. To evaluate their insights, Philippe and his team built a Java prototype that runs on BlackBerry and Android.

Rob Johnson noted that it seems an attacker could game the system by going through a user's address book. Philippe responded that the authentication score may not just rely on phonebook/call history but also GPS, and may also decrease every time the user looks at his call history. Fabian Monrose noted that if someone snatches a phone and uses it immediately, in the absence of biometrics there are very few things this system can do. Tadayoshi Kohno wondered whether their technique can be adopted by Google and others for Web-based authentication. Philippe replied that they collect more dynamic and detailed data as mobile devices, and network providers have a lot more data than a simple user Web access profile. Eric Goldman asked whether a user can still authenticate when the authentication score decreases during the day. There is always the option to log in with a password, which also supercharges the authentication score.

SECURE SYSTEMS AND APPLICATIONS

Summarized by Akshay Dua (dakshay@gmail.com)

- ***Garm: Cross Application Data Provenance and Policy Enforcement***
  Brian Demsky, University of California, Irvine

Brian Demsky presented Garm, a system that can prevent accidental disclosure of arbitrary data and track its history even when used across multiple applications. Brian designed Garm to seamlessly support legacy applications and current data use patterns, such as protecting data even if copied to USB drives.

Garm consists of a dynamic binary-rewriter that instruments binaries under its control to track the provenance (i.e., history) of the application's state during its execution. Further, Garm implements an intermediate layer between the application and the OS to enforce policies for, and possibly encrypt, each byte of data that is read or written by that application. A remote policy server along with a Trusted Platform Module (TPM) on the user's machine are responsible for making sure that Garm and any associated policies have not been tampered with.

Would it be important to track anything other than the sources of input that created the data, such as time? It would be easy to track the time as well. Is such fine-grained control worthwhile? Brian highlighted the advantage of fine-grained control with an email example where different emails in the inbox could potentially have a wide variety of different policies (as selected by the senders). Coarse-grained access control, on the other hand, would cause all policies to apply to all email in the inbox. He also mentioned that there was an opportunity here to optimize, since blocks of bytes would have the same provenance.

- ***Convergence of Desktop and Web Applications on a Multi-Service OS***
  Helen J. Wang, Microsoft Research, Redmond; Alexander Moshchuk, University of Washington, Seattle; Alan Bush, Microsoft Corporation

Alexander Moshchuk spoke about ServiceOS, a new operating system that treats applications as services, thus enabling convergence of Web and desktop applications. Alexander pointed out that the PC sharing model has changed from a multi-user model to a single-user, multi-application model. However, although browsers can support multiple applications, they were not designed to be operating systems for rich applications. The major differences are that browsers do not provide reliable cross-application protection, they have many vulnerabilities, they do not really manage resources such as CPU or network, and they do not provide Web applications with APIs to access devices like cameras on the system.

Alexander also argued that browsers have the right principal model, where the principal is the application rather than the user. Systems where the user is the principal are plagued with malware that can misuse the privileges of the current user. ServiceOS incorporates the best of both worlds: it leverages the principal model used in browsers and combines it with features from a traditional OS. ServiceOS models each application as a service consisting of a chain of content and content renderers (e.g., a movie is rendered by a Python movie player, which in turn is rendered by Jython, which is rendered by the JVM). Each entity in the chain can have different owners and the owner of the head of the chain is the principal. The unit of protection in ServiceOS is the principal and the unit of failure containment and resource allocation is an instance of the principal.

Had they looked into Mobile Agent Systems (MAS), which dealt with many similar issues? They had not looked into that line of research in detail but were confident that their vision of a cache-only Web-centric device was different from the MAS that was being referred to. Another person was concerned about backward compatibility. Alexander said that it was an issue, but suggested that one could start with device classes where it's easy to be backward-compatible and then move on to harder ones. What if the owner wants a user to use a particular renderer for some content? That's a challenging issue that requires carefully defining the precedence between users, the OS, and content providers in terms of who specifies the mapping from content to its renderer.

- ***System Configuration as a Privilege***
  Glenn Wurster and Paul C. van Oorschot, Carleton University

Glenn Wurster talked about creating a system configuration privilege that would be separate from the traditional root. A separate privilege could be used to prevent stealthy configuration changes and to restrict the abilities of installers. Further, he pointed out that systems are normally used for doing work and that configuration is seldom changed.

Glenn emphasized that most install procedures require the user to become superuser first, thus granting the installer unrestricted access to the file system. He mentioned that existing file-system protection mechanisms (e.g., Discretionary or Mandatory Access Control) seem to assume that the system is in a steady state—applications are not being installed or removed. His personal observation is that existing protection mechanisms can handle installs but were not designed for them. Glenn's approach is to create a new configuration privilege and assign it to a single configuration daemon. The privilege cannot be obtained or granted to any other entity. Installers can then interact with the configuration daemon to request configuration changes. However, since it is hard to differentiate between applications that are installers and those that aren't, all applications need to use the configuration daemon to make configuration changes. The configuration daemon rejects or performs configuration changes based on user input, the presence of a specific USB key, file-system state, or other criteria. This approach helps limit dangerous configuration changes regardless of whether or not the application is an installer.

What is the difference between a traditional install and one using this system? Glenn responded that the installers' requests for changes can still be rejected by the configuration daemon (e.g., if the specific USB key is not inserted). Another person suggested an alternative that would involve installing on an overlay file system, verifying the changes, and then applying it to the real file system. Glenn pointed out that for this to work one must know which applications are installers beforehand, but that can be difficult to figure out. Someone else asked how the system deals with trojans or good applications that change the configuration in an unwanted manner. Glenn replied that their system treated the two the same.

writing for *;login:*

Writing is not easy for most of us. Having your writing rejected, for any reason, is no fun at all. The way to get your articles published in *;login:*, with the least effort on your part and on the part of the staff of *;login:*, is to submit a proposal first.

PROPOSALS

In the world of publishing, writing a proposal is nothing new. If you plan on writing a book, you need to write one chapter, a proposed table of contents, and the proposal itself and send the package to a book publisher. Writing the entire book first is asking for rejection, unless you are a well-known, popular writer.

;login: proposals are not like paper submission abstracts. We are not asking you to write a draft of the article as the proposal, but instead to describe the article you wish to write. There are some elements that you will want to include in any proposal:

- What's the topic of the article?
- What type of article is it (case study, tutorial, editorial, mini-paper, etc.)?
- Who is the intended audience (syadmins, programmers, security wonks, network admins, etc.)?
- Why does this article need to be read?
- What, if any, non-text elements (illustrations, code, diagrams, etc.) will be included?
- What is the approximate length of the article?

Start out by answering each of those six questions. In answering the question about length, bear in mind that a page in *;login:* is about 600 words. It is unusual for us to publish a one-page article or one over eight pages in length, but it can happen, and it will, if your article deserves it. We suggest, however, that you try to keep your article between two and five pages, as this matches the attention span of many people.

The answer to the question about why the article needs to be read is the place to wax enthusiastic. We do not want marketing, but your most eloquent explanation of why this article is important to the readership of *;login:*, which is also the membership of USENIX.

UNACCEPTABLE ARTICLES

;login: will not publish certain articles. These include but are not limited to:

- Previously published articles. A piece that has appeared on your own Web server but not been posted to USENET or slashdot is not considered to have been published.
- Marketing pieces of any type. We don't accept articles about products. "Marketing" does not include being enthusiastic about a new tool or software that you can download for free, and you are encouraged to write case studies of hardware or software that you helped install and configure, as long as you are not affiliated with or paid by the company you are writing about.
- Personal attacks

FORMAT

The initial reading of your article will be done by people using UNIX systems. Later phases involve Macs, but please send us text/plain formatted documents for the proposal. Send proposals to login@usenix.org.

DEADLINES

For our publishing deadlines, including the time you can expect to be asked to read proofs of your article, see the online schedule at http://www.usenix.org/publications/login/sched.html.

COPYRIGHT

You own the copyright to your work and grant USENIX permission to publish it in *;login:* and on the Web. USENIX owns the copyright on the collection that is each issue of *;login:*. You have control over who may reprint your text; financial negotiations are a private matter between you and any reprinter.

FOCUS ISSUES

In the past, there has been only one focus issue per year, the December Security edition. In the future, each issue may have one or more suggested focuses, tied either to events that will happen soon after *;login:* has been delivered or events that are summarized in that edition.

HotPar 10

2nd USENIX Workshop on Hot Topics in Parallelism

June 14–15, 2010, Berkeley, CA
Sponsored by USENIX in cooperation with ACM SIGMETRICS, ACM SIGSOFT, ACM SIGOPS, and ACM SIGARCH

HotPar '10 will bring together researchers and practitioners doing innovative work in the area of parallel computing. HotPar recognizes the broad impact of multicore computing and seeks relevant contributions in all fields, including application design, languages and compilers, systems, and architecture.

To ensure a productive workshop environment, attendance will be limited to 75 participants. Each potential participant should submit a position paper of five or fewer pages by January 24, 2010.

USENIX
The Advanced Computing Systems Association

Call for Papers! http://www.usenix.org/hotpar10/cfp

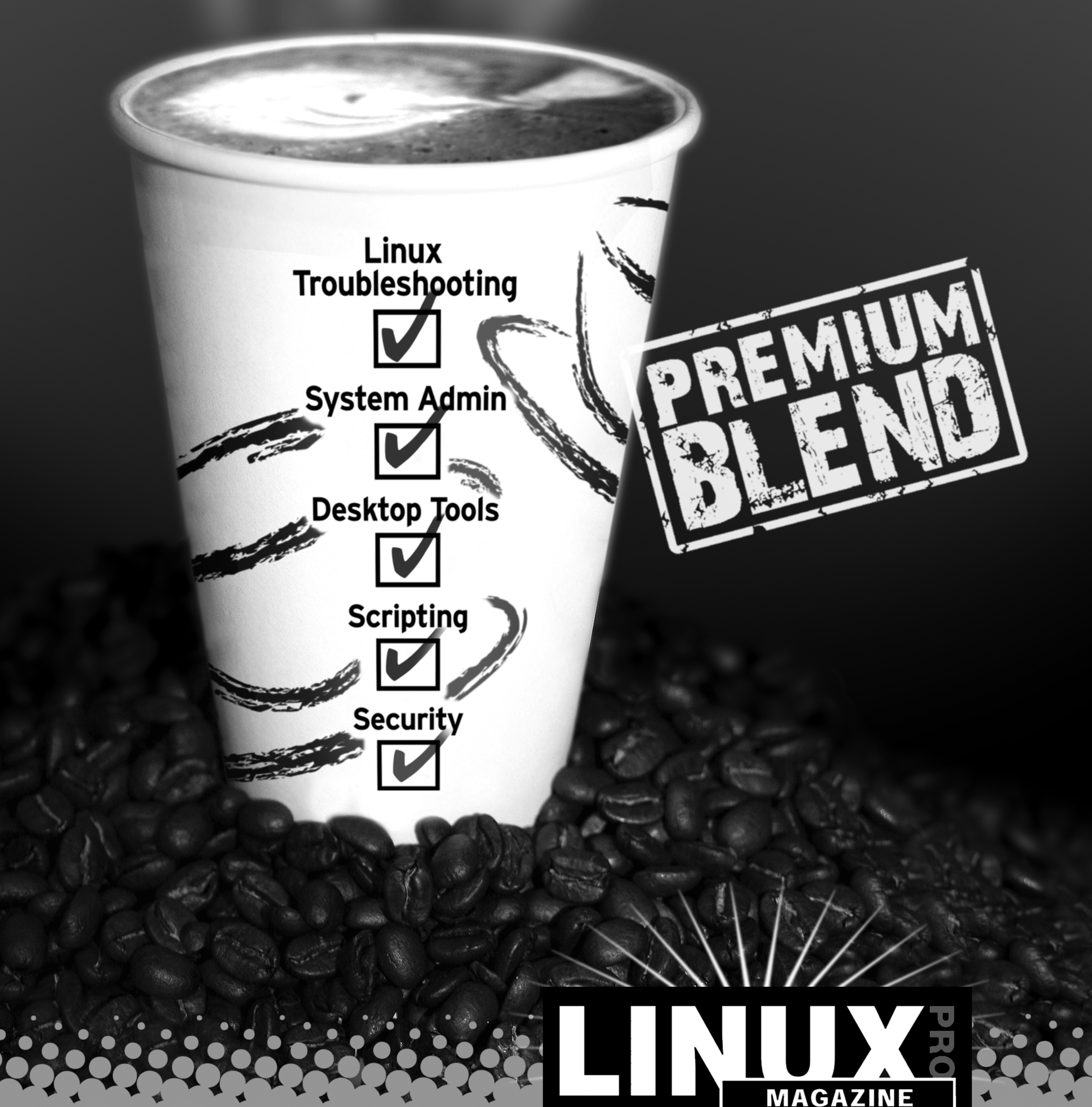
USENIX Technical Conferences Week

June 23–25, 2010 • Boston, MA

USENIX Technical Conferences Week will feature:

- **2010 USENIX Annual Technical Conference (USENIX ATC '10)**
  Submissions are due January 11, 2010; see http://www.usenix.org/atc10/cfp.
- **USENIX Conference on Web Application Development (WebApps '10)**
  Paper titles and abstracts are due January 4, 2010; see http://www.usenix.org/webapps10/cfp.
- **3rd Workshop on Online Social Networks (WOSN 2010)**
  Paper submissions are due February 18, 2010; see http://www.usenix.org/wosn10/cfp.
- **2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)**
- **2nd Workshop on Hot Topics in Storage and File Systems (HotStorage '10)**

USENIX is seeking suggestions and proposals for offerings (e.g., sessions, workshops, tutorials) for co-location. For more information, please contact the USENIX Executive Director, Ellie Young, at ellie@usenix.org.



LINUX PRO MAGAZINE

If you like the taste of Linux, why not treat yourself to the best?

Linux Pro Magazine delivers real-world solutions for the technical reader. In every issue, you'll find advanced techniques for configuring and securing Linux systems. Learn about the latest tools and discover the secrets of the experts in Linux Pro. Each issue includes a full Linux distribution on DVD.

www.linuxpromagazine.com

USENIX

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!

FAST '10

8TH USENIX CONFERENCE ON FILE AND STORAGE TECHNOLOGIES

February 23–26, 2010, San Jose, CA

Join us in San Jose, CA, February 23–26, 2010, for the latest in file and storage technologies. The 8th USENIX Conference on File and Storage Technologies (FAST '10) brings together storage system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems.

Don't miss these co-located workshops, both of which will take place on February 22, 2010:

- First USENIX Workshop on Sustainable Information Technology (SustainIT '10)
- 2nd USENIX Workshop on the Theory and Practice of Provenance (TaPP '10)

Full program info and registration are available at http://www.usenix.org/fast10.

USENIX

http://www.usenix.org/fast10